FORSCHUNGSZENTRUM JÜLICH GmbH
Zentralinstitut für Angewandte Mathematik
D-52425 Jülich, Tel. (02461) 61-6402

Interner Bericht

Beiträge zum Wissenschaftlichen Rechnen
Ergebnisse des
Gaststudentenprogramms 2005
des John von Neumann-Instituts
für Computing

Rüdiger Esser (Hrsg.)

FZJ-ZAM-IB-2005-13

November 2005

(letzte Änderung: 1. 11. 2005)

Vorwort

Die Ausbildung im Wissenschaftlichen Rechnen ist neben der Bereitstellung von Supercomputer-Leistung und der Durchführung eigener Forschung eine der Hauptaufgaben des John von Neumann-Instituts für Computing (NIC) und hiermit des ZAM als wesentlicher Säule des NIC. Um den akademischen Nachwuchs mit verschiedenen Aspekten des Wissenschaftlichen Rechnens vertraut zu machen, führte das ZAM in diesem Jahr zum sechsten Mal während der Sommersemesterferien ein Gaststudentenprogramm durch. Entsprechend dem fächerübergreifenden Charakter des Wissenschaftlichen Rechnens waren Studenten der Natur- und Ingenieurwissenschaften, der Mathematik und Informatik angesprochen. Die Bewerber mussten das Vordiplom abgelegt haben und von einem Professor empfohlen sein.

Die dreizehn vom NIC ausgewählten Teilnehmer kamen für zehn Wochen, vom 8. August bis 14. Oktober 2005, ins Forschungszentrum. Sie beteiligten sich hier an den Forschungs- und Entwicklungsarbeiten des ZAM und der NIC Forschergruppe Computer-gestützte Biologie und Biophysik. Sie wurden jeweils einem oder zwei Wissenschaftler zugeordnet, der mit ihnen zusammen eine Aufgabe festlegte und sie bei der Durchführung anleitete.

Die Gaststudenten und ihre Betreuer waren:

Claudia Albrecht	Wolfgang Meyer
Quang Minh Bui	Marc-André Hermanns
Jens Doleschal	Bernd Mohr
Dirk Elbeshausen	Wolfgang Frings, Sonja Dominiczak
Jens Grieger	Boris Orth
Britta Hennecken	Herwig Zilken
Tobias Hertkorn	Norbert Eicker
Martin Hoch	Bernd Schuller
Christoph Junghans	Ulrich Hansmann, Jan Meinke
Karsten Kahl	Bernhard Steffen
Björn Kuhlmann	Thomas Düssel
Robert Speck	Inge Gutheil
Aiko Voigt	Stefan Krieg

Zu Beginn ihres Aufenthalts erhielten die Gaststudenten eine viertägige Einführung in die Programmierung und Nutzung der Parallelrechner im ZAM. Um den Erfahrungsaustausch untereinander zu fördern, präsentierten die Gaststudenten am Ende ihres Aufenthalts ihre Aufgabenstellung und die erreichten Ergebnisse. Sie verfassten zudem Beiträge mit den Ergebnissen für diesen Internen Bericht des ZAM.

Wir danken den Teilnehmern für ihre engagierte Mitarbeit - schließlich haben sie geholfen, einige aktuelle Forschungsarbeiten weiterzubringen - und den Betreuern, die tatkräftige Unterstützung dabei geleistet haben, insbesondere Bernd Mohr und Boris Orth, die den Einführungskurs gehalten haben.

Ebenso danken wir allen, die im ZAM und der Verwaltung des Forschungszentrums bei Organisation und Durchführung des diesjährigen Gaststudentenprogramms mitgewirkt haben. Besonders hervorzuheben ist die finanzielle Unterstützung durch den Verein der Freunde und Förderer des FZJ und die Firma IBM.
Es ist beabsichtigt, das erfolgreiche Programm künftig fortzusetzen, schließlich ist die Förderung des wissenschaftlichen Nachwuchses dem Forschungszentrum ein besonderes Anliegen.

Weitere Informationen über das Gaststudentenprogramm, auch die Ankündigung für das kommende Jahr, findet man unter http://www.fz-juelich.de/zam/gaststudenten.

Jülich, November 2005 Rüdiger Esser

Inhalt

Automatisierte Daten-Transformation im GALA-Projekt

Claudia Albrecht

Technische Universität Dresden
Fakultät Mathematik und Naturwissenschaften
Fachrichtung Mathematik

E-mail: s9458785@rcs.urz.tu-dresden.de

Zusammenfassung:

Heutzutage werden immer größere Datenbestände bei der Planung neuer oder der Verbesserung bereits vorhandener Produkte in der Industrie erhoben. Eine wichtige Aufgabe besteht nun darin, die verborgenen Informationen innerhalb dieser Daten zu erkennen und auszuwerten.[7] Im Rahmen des GALA-Projektes (Grünenthal Applied Life Science Analysis) wurden für die automatisierte Datenauswertung Programme zur Transformation von Originaldaten implementiert. Eine sehr nützliche und häufig angewendete Familie ist die der Box-Cox-Transformation.

Einleitung

Viele statistische Auswertemethoden/Verfahren stellen gewisse Anforderungen an die Verteilung von Zufallsvariablen, wie z.B. die Normalverteilung oder Linearität der Daten. In der Praxis findet man jedoch meistens Verteilungen, die nicht einmal näherungsweise durch die Normalverteilung beschrieben werden können. Daher stellt sich die Frage, welche Verfahren ohne die Normalverteilungsannahme möglich sind oder welche Transformation mit Annäherung an die Normalverteilung eventuell zum Ziel führt.
Hier bieten sich z.B. Transformationen an, welche die Schiefe der Verteilung mittels nichtlinearer Transformation beseitigen. Diese Symmetrisierung dient der besseren Beurteilung von untypischen Werten (Ausreißern) sowie der Zusammenführung der Lageparameter Modus, Median und Mittelwert, die bei nichtsymmetrischen Verteilungen voneinander abweichen. Transformationen können zudem varianzstabilisierend wirken.

Das folgende Beispiel zeigt die Bedeutung einer Transformation. In der Abbildung 1 findet man den Probability Plot einer Variable. Die Schiefe dieser Variable beträgt 9.2646 und liefert mit einem Verfahren für die Ausreißeridentifizierung, welches auf der Normalverteilung basiert, 4 Ausreißer.
Ausreißer können ein wirkliches Problem darstellen. Gibt es Punkte, die in einer gewissen Entfernung vom Hauptteil der Daten liegen, könnten diese einen übermäßigen Einfluss auf die Schätzungen ausüben. Die Identifizierung von Ausreißern kann durch Tests erfolgen, welche als Annahme mit der Normalverteilung arbeiten. Jedoch kann dadurch bei nichtnormalen Verteilungen ein Messwert als Ausreißer deklariert werden, welcher in Wirklichkeit keiner ist.
Hier ist eine Transformation sinnvoll, welche annähernd zur Normalverteilung führt und damit den Annahmen für die Ausreißertests gerecht wird. Es wurden in dem Beispiel aus Abbildung 1 nach der Logarithmus-Transformation der Variable keine Ausreißer mehr entdeckt. Die Schiefe wurde reduziert und hat jetzt einen Wert von 0.6019. Dies bedeutet, dass die transformierten Daten viel symmetrischer und normalverteilter sind als die Originaldaten. Am Ende des Kapitels Box-Cox-Transformation wird

auf dieses Beispiel noch einmal eingegangen.

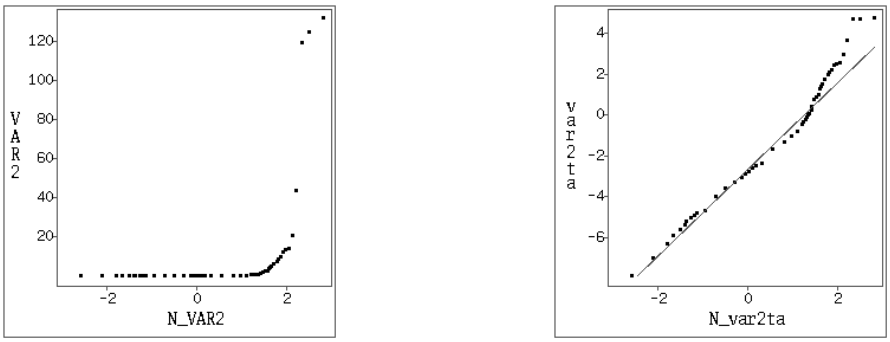


Abbildung 1: Auf der linken Seite ist der Probability Plot der Originaldaten einer Variable, auf der rechten Seite befindet sich der Plot der logarithmisch transformierten Daten

In den folgenden Abschnitten werden einige spezielle Transformations-Familien, insbesondere die Box-Cox-Transformation, vorgestellt. Diese Verfahren wurden im Rahmen dieser Arbeit in der Programmiersprache Fortran95 implementiert.

Box-Cox-Transformation

Dieses Kapitel beschäftigt sich mit der Umwandlung der Daten mit Hilfe einer mathematischen Funktion. Es stellt sich also die Frage, welche Transformation sinnvoll ist. Die Wahl hängt von der Verteilung der Daten ab. Es gibt verschiedene Familien, die sich als sinnvoll erwiesen haben.

Eine nützliche Familie für positive Daten ist die Potenztransformation $y = x^\lambda$. Eine Schwierigkeit wird sichtbar, wenn $\lambda = 0$ und damit $y = x^0 = 1$, d.h. für alle Daten x ist die Transformierte gleich. Aufgrunddessen benutzt man die folgende Potenztransformation $T_P(x)$.

$$T_P(x) = \begin{cases} x^\lambda & \text{für } \lambda \neq 0 \\ \log(x) & \text{für } \lambda = 0 \end{cases} \tag{1}$$

Diese besondere Familie wurde im Detail studiert von Tukey (1957) für $|\lambda| \leq 1$. Sie schließt die bekannte Logarithmus-, die Wurzel- und die inverse Transformation mit ein. Im folgenden wird mit $\log x$ der natürliche Logarithmus (zur Basis e) bezeichnet.
Für $\lambda < 1$ werden kleine Werte gestreckt und große Werte gestaucht, während für $\lambda > 1$ eine stärkere Ausdehnung von großen Werten erfolgt. Es kommt zu einer symmetrischeren Verteilung.

Um die Unstetigkeit bei $\lambda = 0$ zu vermeiden modifizierten Box-Cox (1964) die Potenztransformation:

$$T_{P^*}(x) = x^{(\lambda)} = g_\lambda(x) = \begin{cases} \frac{x^\lambda - 1}{\lambda} & \text{für } \lambda \neq 0 \\ \log(x) & \text{für } \lambda = 0 \end{cases} \tag{2}$$

Diese Transformation dient der Symmetrisierung der Verteilung und der Varianzstabilisierung. Die Box-Cox-Transformation hat sich für viele Anwendungsfälle bewährt.

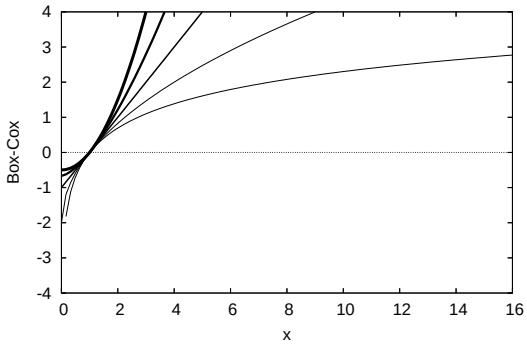


Abbildung 2: Die Box-Cox-Transformation für bestimmte $\lambda = 0.0; 0.5; 1.0; 1.5$ und 2.0 (x=Originaldaten; y=transformierte Werte)

Diese Form der Transformation weist für die graphische Darstellung und den Vergleich drei Vorteile auf:

1. Die Kurven sind monoton wachsend, so dass für jedes λ, $g_\lambda(x)$ die Reihenfolge/Ordnung der Daten, welche transformiert werden, bewahrt.
2. Die Kurven haben für alle λ den Punkt (1,0) gemeinsam.
3. Die Kurven stimmen in der Nähe des Punktes (1,0) nahezu überein, d.h. sie haben eine gemeinsame Tangente in diesem Punkt.

Beweise [6]:

1. Die Ableitung von $g_\lambda(x)$ ist $\frac{dg_\lambda(x)}{dx} = g'_\lambda(x) = x^{(\lambda-1)}$ für alle λ, einschließlich 0. Da $g'_\lambda(x)$ existiert und positiv ist für alle $x > 0$, ist $g_\lambda(x)$ eine monoton wachsende stetige Funktion von x für alle λ.
2. Für jedes λ, einschließlich $\lambda = 0$, gilt: $g_\lambda(1) = 0$.
3. Es gilt $g'_\lambda(1) = 1$, damit ist der Anstieg der Tangente an jeder Kurve im Punkt ($x := 1$) gleich eins. Folglich haben die Kurven die gleiche Tangente. Die Ableitung $g'_\lambda(x)$ ist ebenfalls stetig für alle positiven x. Weil $g'_\lambda(x)$ für x in der Nähe von 1 nahe bei 1 liegt, haben die Kurven für x-Werte in der Nähe von 1 fast die gleiche Steigung.

Es soll weiter gezeigt werden, dass für jedes x, $g_\lambda(x) = y(\lambda, x)$, $g'_\lambda(x)$ usw. stetige Funktionen von λ sind [6].
Sei $\lambda \neq 0$. Da $(x^\lambda - 1)$ und λ stetige Funktionen von λ sind, ist der Quotient dann ebenfalls stetig bei $\lambda \neq 0$.
Für $\lambda = 0$ hat der Quotient eine undefinierte Form, aber es kann mit einer Standardmethode behandelt werden. Zuerst sei $x^\lambda = e^{\lambda \log(x)}$, und die Ableitung bezüglich λ von diesem Ausdruck ist $e^{\lambda \log(x)} * \log(x)$. Da Zähler und Nenner des Grenzwertes

$$\lim_{\lambda \to 0} y(\lambda, x) = \lim_{\lambda \to 0} \left(\frac{e^{\lambda \log(x)} - 1}{\lambda} \right)$$

gegen Null konvergieren, wird der Satz von l'Hospital angewendet:

$$\lim_{\lambda \to 0} y(\lambda, x) = \lim_{\lambda \to 0} \left(\frac{e^{\lambda \log(x)} * \log(x)}{1} \right) = \log(x).$$

Dieses Resultat zeigt, dass

$$\lim_{\lambda \to 0} y(\lambda, x) = y(0, x).$$

Daraus folgt aufgrund der Definition, dass $y(\lambda, x)$ in $\lambda = 0$ für jedes positive x stetig ist.
Zusammenfassend kann gesagt werden: $g_\lambda(x)$ *ist eine Familie von Funktionen, die stetig vom Parameter* λ *indiziert wird.*
Der Beweis für die Stetigkeit von $g'_\lambda(x), g''_\lambda(x)$, erfolgt ähnlich.

Es wird vorausgesetzt, dass die transformierten Beobachtungen $x^{(\lambda)}$ unabhängig und identisch $N(\mu, \sigma^2)$-verteilt sind. Schätzwerte für λ, μ, σ^2 ergeben sich durch die Maximierung der Likelihoodfunktion. Wie später gezeigt wird, ist dann die logarithmierte Likelihoodfunktion der Beobachtungen, bis auf eine Konstante

$$logL_{max}(\lambda) = -\frac{n}{2} \log \hat{\sigma}^2(\lambda) + (\lambda - 1) \sum_{i=1}^{n} \log x_i.$$

Ein numerisches Suchverfahren wird den maximierenden Wert für lambda finden, oder zumindest einen guten Näherungswert desselben. In vielen Fällen sehen die Daten nach der Transformation normalverteilter aus als davor.

Jedoch ist die Box-Cox-Transformation kein Allheilmittel für alle Fälle, insbesondere bei Stichprobenverteilungen mit mehreren Modalwerten erreicht man durch Transformation häufig keine Verbesserung.

Herleitung der Likelihood-Funktion

Als erstes wird ein zugrundeliegendes mathematisches Modell vorgestellt. Es bezeichne X eine Zufallsvariable (Zufallsgröße), die auf einem geeigneten Wahrscheinlichkeitsraum $(\Omega, \mathcal{A}, \mathbb{P})$ als eine Abbildung von Ω in die Menge der reellen Zahlen definiert ist, und x bezeichnet im Folgenden eine Realisation der Zufallsvariable X. Des Weiteren sei

$$F : \mathbb{R} \to [0; 1] \quad \text{mit} \quad F(x) := \mathbb{P}(X \leq x) \quad \forall x \in \mathbb{R}$$

die Verteilungsfunktion der Zufallsvariable X mit den bekannten Eigenschaften.

Eine Stichprobenfunktion, deren Wert einen unbekannten Parameter der den Beobachtungen zugrundeliegenden Verteilung schätzt, heißt Schätzer bzw. eine Schätzfunktion. Seien $X_1, ..., X_n$ (n sei die Anzahl der Originaldaten) unabhängig und identisch wie X verteilte Zufallsgrößen. Seien $Y_1, ..., Y_n$ die unabhängigen und identisch verteilten Zufallsgrößen der transformierten Daten mit Verteilungsfunktion F_Y. Außerdem sei vorausgesetzt, dass die Verteilungsfunktion F_Y eine Dichtefunktion $f_Y(x)$ besitze. Wenn wir annehmen, dass die transformierten Beobachtungen (Gleichung 1) $x^{(\lambda)} = g_\lambda(x) \sim N_1(\mu, \sigma^2)$-verteilt sind, dann ist die Verteilungsfunktion und die Dichtefunktion für die transformierten Daten gegeben mit:

$$\begin{aligned} \mathbb{P}(Y \leq y) = F_Y(y) &= \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{y} e^{-\frac{(t-\mu)^2}{2\sigma^2}} \, \mathrm{d}t \\ f_Y(y) &= \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y-\mu)^2}{2\sigma^2}} \end{aligned}$$

Da die Funktion g_λ auf $\mathbb{R}_+ \; \forall \lambda \in \mathbb{R}$ streng monoton wachsend ist, ergibt sich die Verteilungsfunktion für X zu

$$\begin{aligned} F_X(x) &= \mathbb{P}(X \leq x) = \mathbb{P}\left(g_\lambda(X) \leq g_\lambda(x)\right) \\ &= \mathbb{P}\left(Y \leq g_\lambda(x)\right) = F_Y\left(g_\lambda(x)\right). \end{aligned}$$

Hieraus ergibt sich für die Dichte von X

$$\begin{aligned} f_X(x) &= \frac{d}{dx}F_X(x) = \frac{d}{dx}F_Y\left(g_\lambda(x)\right) \\ &= f_Y\left(g_\lambda(x)\right) * g'_\lambda(x). \end{aligned}$$

Da

$$g'_\lambda(x) = \left\{ \begin{array}{cc} \left(\lambda x^{(\lambda-1)}\right)/\lambda & \text{für } \lambda \neq 0 \\ 1/x & \text{für } \lambda = 0 \end{array} \right\} = x^{\lambda-1}$$

ergibt sich

$$f_X(x) = f(x;\mu,\sigma,\lambda) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x^{(\lambda)}-\mu)^2}{2\sigma^2}} * x^{\lambda-1}.$$

Die Likelihood-Funktion der Stichprobe $x_1, ..., x_n$ ist definiert als

$$L(\mu,\sigma,\lambda) = \prod_{i=1}^{n} f(x_i;\mu,\sigma,\lambda).$$

Die Maximum-Likelihood-Methode ergibt als Schätzer für das unbekannte λ jeden Wert $\hat{\lambda}$, der die Likelihood-Funktion über λ maximiert. Es folgt die Ermittlung der Likelihood-Funktion.

$$\begin{aligned} L(\mu,\sigma,\lambda) &:= \left(\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x_1^{(\lambda)}-\mu)^2}{2\sigma^2}} * x_1^{\lambda-1}\right) * ... * \left(\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x_n^{(\lambda)}-\mu)^2}{2\sigma^2}} * x_n^{\lambda-1}\right) \\ &= \left(\frac{1}{(\sqrt{2\pi} * \sigma)^n}\right) * e^{-\left(\frac{(x_1^{(\lambda)}-\mu)^2+...+(x_n^{(\lambda)}-\mu)^2}{2\sigma^2}\right)} * \prod_{i=1}^{n} x_i^{\lambda-1} \\ &= \left(2\pi * \sigma^2\right)^{-\frac{n}{2}} * e^{-\sum_{i=1}^{n} \frac{(x_i^{(\lambda)}-\mu)^2}{2\sigma^2}} * \prod_{i=1}^{n} x_i^{\lambda-1} \end{aligned}$$

Dabei stellt es eine Erleichterung dar, wenn man zur logarithmierten Likelihood-Funktion übergeht und so das Produkt durch eine Summe ersetzen kann.

$$\begin{aligned} \log L(\mu,\sigma,\lambda) &= \log\left(2\pi * \sigma^2\right)^{-\frac{n}{2}} + \log\left(e^{-\sum_{i=1}^{n} \frac{(x_i^{(\lambda)}-\mu)^2}{2\sigma^2}}\right) + \log \prod_{i=1}^{n} x_i^{\lambda-1} \\ &= -\frac{n}{2}\log 2\pi - \frac{n}{2}\log\sigma^2 - \sum_{i=1}^{n} \frac{(x_i^{(\lambda)}-\mu)^2}{2\sigma^2} + \sum_{i=1}^{n} \log x_i^{\lambda-1} \\ &= -\frac{n}{2}\log 2\pi - \frac{n}{2}\log\sigma^2 - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i^{(\lambda)}-\mu)^2 + (\lambda-1)\sum_{i=1}^{n}\log x_i \end{aligned}$$

Die Maximum Likelihood-Schätzungen von μ und σ^2 für ein gegebenes λ sind die klassischen Schätzer der Normalverteilung.

$$\begin{aligned} \frac{\partial \log L(\mu,\sigma,\lambda)}{\partial\mu} &= -\frac{1}{2\sigma^2}\sum_{i=1}^{n} 2(x_i^{(\lambda)}-\mu)(-1) \\ &= \frac{1}{\sigma^2}\sum_{i=1}^{n}(x_i^{(\lambda)}-\mu) \stackrel{!}{=} 0 \\ 0 = \sum_{i=1}^{n}(x_i^{(\lambda)}-\mu) &= \sum_{i=1}^{n} x_i^{(\lambda)} - \sum_{i=1}^{n}\mu = \sum_{i=1}^{n} x_i^{(\lambda)} + n\mu \\ \Rightarrow \hat{\mu} &= \frac{1}{n}\sum_{i=1}^{n} x_i^{(\lambda)} \end{aligned}$$

$$\begin{aligned} \frac{\partial \log L(\mu,\sigma,\lambda)}{\partial\sigma} &= -\frac{n}{\sigma} - \frac{1}{2} * (-2) * \sigma^{-3} * \sum_{i=1}^{n}(x_i^{(\lambda)}-\mu)^2 \\ &= \frac{1}{\sigma^3}\sum_{i=1}^{n}(x_i^{(\lambda)}-\mu)^2 - \frac{n}{\sigma} \stackrel{!}{=} 0 \\ n &= \frac{1}{\sigma^2}\sum_{i=1}^{n}(x_i^{(\lambda)}-\mu)^2 \\ \Rightarrow \hat{\sigma}^2 &= \frac{1}{n}\sum_{i=1}^{n}(x_i^{(\lambda)}-\mu)^2 \end{aligned}$$

Einsetzen der Likelihood-Schätzer in die obige logarithmierte Likelihood-Funktion ergibt die folgende Zielfunktion:

$$\begin{aligned}
\log L(\hat{\mu},\hat{\sigma},\lambda) &= -\frac{n}{2}\log 2\pi - \frac{n}{2}\log\hat{\sigma}^2 - \frac{1}{2\hat{\sigma}^2}\sum_{i=1}^{n}(x_i^{(\lambda)}-\hat{\mu})^2 + (\lambda-1)\sum_{i=1}^{n}\log x_i \\
&= -\frac{n}{2}\log\hat{\sigma^2} + (\lambda-1)\sum_{i=1}^{n}\log x_i - \frac{n}{2}\log 2\pi - \frac{1}{2}\frac{n}{\sum_{i=1}^{n}(x_i^{(\lambda)}-\hat{\mu})^2} * \sum_{i=1}^{n}(x_i^{(\lambda)}-\hat{\mu})^2 \\
&= -\frac{n}{2}\log\left[\frac{1}{n}\sum_{i=1}^{n}(x_i^{(\lambda)}-\hat{\mu})^2\right] + (\lambda-1)\sum_{i=1}^{n}\log x_i - \frac{n}{2}\log 2\pi - \frac{n}{2} \\
&= -\frac{n}{2}\log\left[\frac{1}{n}\sum_{i=1}^{n}(x_i^{(\lambda)}-\hat{\mu})^2\right] + (\lambda-1)\sum_{i=1}^{n}\log x_i - \frac{n}{2}(\log 2\pi + 1)
\end{aligned}$$

Für die Maximierung wird eine Nullstelle der ersten Ableitung gesucht, d.h. die Gleichung

$$\frac{\delta log L(\hat{\mu},\hat{\sigma},\lambda)}{\delta\lambda} \stackrel{!}{=} 0 \tag{3}$$

ist zu lösen. Gesucht ist ein λ-Wert, der eine möglichst gute Anpassung an die Normalverteilung liefert. Der letzte Term in $\log L(\hat{\mu},\hat{\sigma},\lambda)$ ist unabhängig von $\hat{\sigma},\hat{\mu}$ und von der Stichprobe $(x_1,x_2,...,x_n)$. Er kann bei der Maximierung über λ damit unbeachtet bleiben.
Es ergibt sich somit die zu maximierende Funktion zu:

$$\log L_{max}(\lambda) = -\frac{n}{2}\log\hat{\sigma}^2(\lambda) + (\lambda-1)\sum_{i=1}^{n}\log x_i \tag{4}$$

Die Ausführung der Maximierung ist durch das Benutzen eines numerischen Verfahren möglich, welches Gleichung (3) iterativ löst, oder mit einem einfachen Plot von $\log(L_{max}(\lambda))$ gegen λ. Ein Plot ist allemal nützlich, um das lokale Verhalten von L_{max} in der Nachbarschaft von $\hat{\lambda}$ zu beobachten [2].

Ermittlung des λ-Wertes mit Hilfe eines Plots

Für feste λ-Werte wird die logarithmierte Likelihood-Funktion berechnet. Im speziellen waren dies $\lambda = -2,-1,-\frac{1}{2},-\frac{1}{3},-\frac{1}{4},-\frac{1}{5},-\frac{1}{6},0,\frac{1}{6},\frac{1}{5},\frac{1}{4},\frac{1}{3},\frac{1}{2},1,2$.
Zu beachten ist, dass bei einem $\lambda \neq 0$ Auslöschung auftreten kann. Dies bedeutet, dass eventuell der Zähler der Formel (2) ($e^{\lambda\log(x)} - 1$) nahe bei 0 liegt. Es ist somit zu überprüfen, ob $e^{\lambda\log(x)}$ dicht bei 1 liegt und damit $\lambda\log(x)$ sich in der Nähe von Null befindet. Es wurde ein Kriterium integriert, das in dem Fall von $|\lambda\log(x_i)| \leq 10^{-7}, i \in [1,n]$, nicht die Transformierte durch den Quotienten, sondern mit $\log(x_i)$ bestimmt.

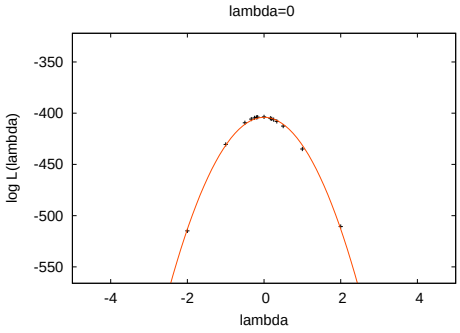


Abbildung 3: Die logarithmierte Likelihood-Funktion für ausgewählte λ-Werte
$\lambda = -2,-1,-\frac{1}{2},-\frac{1}{3},-\frac{1}{4},-\frac{1}{5},-\frac{1}{6},0,\frac{1}{6},\frac{1}{5},\frac{1}{4},\frac{1}{3},\frac{1}{2},1,2$

Das Maximum der 15 log-Likelihood-Funktionswerte liefert dann das gesuchte $\hat{\lambda}$ für die Transformation der Originaldaten mit einer näherungsweisen Normalverteilung der transformierten Daten.

Ermittlung des λ-Wertes iterativ

Bei der iterativen Methode für die Bestimmung des $\hat{\lambda}$-Wertes geht es um die Lösung der Gleichung (3) mit Hilfe des Newton-Verfahrens:

$$\max_{\lambda} \quad \log L_{max}(\lambda) \rightarrow \hat{\lambda} \Rightarrow \frac{\delta log L_{max}(\lambda)}{\delta \lambda} = f(\lambda) \stackrel{!}{=} 0$$

Es muss also eine Nullstellenberechnung von $f(\lambda)$, d.h. der ersten Ableitung der logarithmierten Likelihoodfunktion in Abhängigkeit von λ, stattfinden.
Für ein beliebiges Element $\lambda_0 \in \mathbb{R}$ vermutet man, dass die Elemente $f(\lambda_0) = f(\lambda_0) - f(\hat{\lambda})$ und $f'(\lambda_0)(\lambda_0 - \hat{\lambda})$ „nahe " beieinander liegen und demzufolge die Lösung der linearen Gleichung, also $\lambda_1 = \lambda_0 - [f'(\lambda_0)]^{-1} f(\lambda_0)$, das gesuchte Element $\hat{\lambda}$ approximiert. Auf diese Weise konstruiert man, ausgehend von λ_0, die sogenannte NEWTONsche Näherungsfolge

$$\lambda_{n+1} = \lambda_n - \frac{f(\lambda_n)}{f'(\lambda_n)} \qquad (\text{ n = 0, 1,... }). \tag{5}$$

Es wird solange ein weiteres λ_{n+1} berechnet, bis ein Abbruchkriterium erfüllt ist.
Es sollte ein Startwert gewählt werden, welcher relativ nahe bei dem Endergebnis $\hat{\lambda}$ liegt.

Die Formel für die erste und zweite Ableitung der logarithmierten Likelihood-Funktion sind:

$$\begin{aligned}
\log L_{max}(\lambda) &= -\frac{n}{2} \log \hat{\sigma}^2(\lambda) + (\lambda - 1) \sum_{i=1}^{n} \log x_i \\
f(\lambda) &= \frac{\delta \log L_{max}(\lambda)}{\delta \lambda} = -\frac{n * \left(\hat{\sigma}^2(\lambda)\right)'}{2 * \hat{\sigma}^2(\lambda)} + \sum_{i=1}^{n} \log x_i \\
f'(\lambda) &= \frac{\delta^2 \log L_{max}(\lambda)}{\delta \lambda^2} = -\frac{n}{2} \left(\frac{\left(\hat{\sigma}^2(\lambda)\right)'' \hat{\sigma}^2(\lambda) - \left(\left(\hat{\sigma}^2(\lambda)\right)'\right)^2}{\left(\hat{\sigma}^2(\lambda)\right)^2} \right)
\end{aligned}$$

Die Funktionen $f(\lambda)$ und $f'(\lambda)$ sind demzufolge von λ nur über $\hat{\sigma}^2(\lambda)$ und deren Ableitungen abhängig. Es sei $g_\lambda(x_i) = y(\lambda, x_i)$.

$$\begin{aligned}
\hat{\sigma}^2(\lambda) &= \frac{1}{n} \sum_{i=1}^{n} \left(g_\lambda(x_i) - \hat{\mu}(\lambda)\right)^2 \\
\left(\hat{\sigma}^2(\lambda)\right)' &= \frac{2}{n} \sum_{i=1}^{n} \left((g_\lambda(x_i) - \hat{\mu}(\lambda)) * \left(\frac{\delta}{\delta \lambda} y(\lambda, x_i) - \hat{\mu}'(\lambda) \right) \right) \\
\left(\hat{\sigma}^2(\lambda)\right)'' &= \frac{2}{n} \sum_{i=1}^{n} \left(\left(\frac{\delta}{\delta \lambda} y(\lambda, x_i) - \hat{\mu}'(\lambda) \right)^2 + (g_\lambda(x_i) - \hat{\mu}(\lambda)) * \left(\frac{\delta^2}{\delta \lambda^2} y(\lambda, x_i) - \hat{\mu}''(\lambda) \right) \right)
\end{aligned}$$

Als letztes muss also noch auf die Ableitungen der Transformations-Funktion (2) und der Mittelwertfunktion $\hat{\mu}(\lambda)$ eingegangen werden. Auf die detaillierte Herleitung soll hier verzichtet werden.

$$\begin{aligned}
\hat{\mu}(\lambda) &= \frac{1}{n} \sum_{i=1}^{n} g_\lambda(x_i) \\
\hat{\mu}'(\lambda) &= \frac{1}{n} \sum_{i=1}^{n} \frac{\delta}{\delta \lambda} y(\lambda, x_i) \\
\hat{\mu}''(\lambda) &= \frac{1}{n} \sum_{i=1}^{n} \frac{\delta^2}{\delta \lambda^2} y(\lambda, x_i)
\end{aligned}$$

$$
\begin{aligned}
g_\lambda(x_i) &= \begin{cases} \frac{x_i^\lambda - 1}{\lambda} & \text{für } \lambda \neq 0 \\ \log(x_i) & \text{für } \lambda = 0 \end{cases} \\
\frac{\delta}{\delta\lambda} y(\lambda, x_i) &= \begin{cases} \frac{x_i^\lambda(\log(x_i)\lambda - 1) + 1}{\lambda^2} & \text{für } \lambda \neq 0 \\ \frac{\log^2(x_i)}{2} & \text{für } \lambda = 0 \end{cases} \\
\frac{\delta^2}{\delta\lambda^2} y(\lambda, x_i) &= \begin{cases} \frac{\lambda^2 x_i^\lambda (\log(x_i))^2 - 2\left(\lambda x_i^\lambda \log(x_i) - x_i^\lambda + 1\right)}{\lambda^3} & \text{für } \lambda \neq 0 \\ \frac{\log^3(x_i)}{3} & \text{für } \lambda = 0 \end{cases}
\end{aligned}
$$

Es ist zu beachten, dass bei $(\lambda \log x_i) \approx 10^{-k}$ im Zähler von g_λ an k Stellen Auslöschung auftritt. In der ersten Ableitung sind es 2k Stellen und bei $\frac{\delta^2}{\delta\lambda^2} y(\lambda, x_i)$ 3k Stellen Auslöschung. Daher wurde für kleine $(\lambda \log x_i)$-Werte der Zähler von $\frac{\delta^i}{\delta\lambda^i} y(\lambda, x_i)$ durch das (i+1)-te Taylorpolynom $\left(\frac{(\lambda \log x_i)^{(i+1)}}{(i+1)}\right)$ approximiert. Dieses führte für $\frac{\delta^i}{\delta\lambda^i} y(\lambda, x_i)$ auf die Approximation: $\frac{\delta^i}{\delta\lambda^i} y(\lambda, x_i) \approx \frac{\delta^i}{\delta\lambda^i} y(0, x_i)$.

Als Abbruchbedingung wird $|\lambda_{n+1} - \lambda_n| \leq$ 1.0E-8 verwendet.

Konfidenzintervall

Der approximierte $100(1-\alpha)\%$ Konfidenz-Bereich für den wahren Wert von λ, ergibt sich aus [2] :

$$L_{max}(\hat{\lambda}) - L_{max}(\lambda) \leq \frac{1}{2}\, \chi^2_{1,\alpha}$$

und ist die Menge aller λ für die gilt: $L_{max}(\lambda) \in [L_{max}(\hat{\lambda}) - \frac{1}{2}\chi^2_{1,\alpha}; L_{max}(\hat{\lambda}) + \frac{1}{2}\chi^2_{1,\alpha}]$.
Im Plot von $L_{max}(\lambda)$wird der Konfidenzbereich dargestellt durch eine horizontale Linie mit der Höhe $L_{max}(\hat{\lambda}) - \frac{1}{2}\, \chi^2_1(1-\alpha)$ auf der vertikalen Skala. Diese würde die Kurve an zwei Stellen schneiden, deren zugehörige λ-Werte die Endpunkte des approximativen Konfidenzintervalls bilden [3].
Außerdem werden die Konfidenzgrenzen durch die iterative Lösung der Gleichung

$$L_{max}(\lambda) - c \;=\; 0$$

wobei $L_{max}(\hat{\lambda}) - \frac{1}{2}\chi^2_{1,\alpha} = c$ ist, mit Hilfe des Newton-Verfahrens bestimmt.
Dabei kommt es auf die Startwerte für die jeweilige obere und untere Intervallgrenze an. Für die obere Grenze, wird das vorher berechnete $\hat{\lambda}$ auf eine ganze Zahl aufgerundet und für die untere Grenze auf eine ganze Zahl abgerundet.
Implementiert wurde eine statistische Sicherheit von 95%, d.h. α =5% und $\frac{1}{2}\, \chi^2_1(1-\alpha) = 1.92$. Das bedeutet, das Konfidenzintervall enthält den tatsächlichen λ-Wert mit einer Wahrscheinlichkeit von 95%.

Ermittlung der transformierten Daten

Als abschließende Transformation der Daten für eine Weiterverarbeitung kann sowohl die reine Potenztransformation $T_P(x)$ (Gleichung 1) als auch die nach Box-Cox modifizierte Potenztransformation T_{P^*} (Gleichung 2) gewählt werden. Im Falle der Logarithmierung der Daten, ist die Basis des Logarithmus beliebig wählbar. Die möglichen Transformationen unterscheiden sich nur durch eine Ursprungsverschiebung und unterschiedliche Skalierungen, welche aber die grundlegenden Eigenschaften der folgenden Analyse nicht beeinflussen.
Zu beachten ist, dass auch das „beste“ λ nicht unbedingt nützliche transformierte Daten in der Praxis garantiert.

Ein weiterer Betrachtungspunkt ist das Verwenden des $\hat{\lambda}$. Typischerweise würde man nicht den exakten ML-Schätzer von λ in der folgenden Transformation benutzen, sondern den nächsten „günstigen“ Wert, welcher innerhalb des Konfidenzintervalls liegt. So ist z.B. $\hat{\lambda} = 0.11$ und für die Berechnung wird $\lambda = 0$ verwendet. Es gibt aber beträchtlichen Freiraum für eine persönliche Entscheidung in der Wahl von λ. [3]

In dieser Arbeit wurde für die Endtransformation die Formel (1) der einfachen Potenztransformation gewählt und vorerst noch der tatsächliche ML-Schätzer für die Berechnung verwendet.

Beispiel

In dem Eingangsbeispiel (Abbildung 2) sind die Originaldaten der Variable stark rechtsschief. (Schiefe: 9.2646 ; Kurtosis: 88.0515) Aufgrunddessen ist die Box-Cox-Transformation für die Symmetrisierung der Daten sinnvoll.

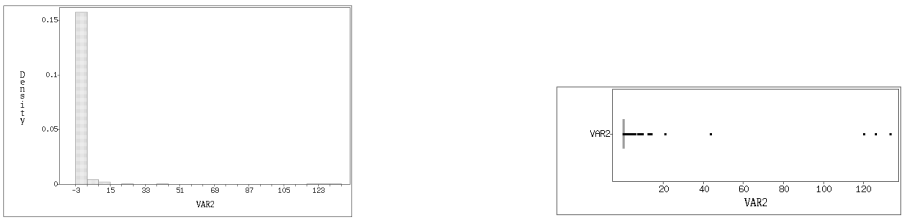


Abbildung 4: links: Histogramm-Plot der Originaldaten, rechts: Box-Plot jener Daten

Aufgrund der für den Plot der logarithmierten Likelihoodfunktion berechneten Werte, ergibt sich als Maximalstelle $\lambda = 0$, folglich die Logarithmus-Transformation. Wie in der Einleitung erwähnt, ergibt sich dann eine Schiefe von 0.6019 für die logarithmierten Daten (Kurtosis: 1.3395). Die iterative Methode liefert als Lösung $\lambda = -0.070$, und ergibt nach der Transformation eine Schiefe von 0.0262 (Kurtosis: 0.6864). Die schon kleinen Schiefe- und Kurtosis-Werte der Verteilung nach der Logarithmus-Transformation konnten also nochmals verringert werden.
Das Konfidenzintervall bei einer statistischen Genauigkeit von 95% ist [-0.110; -0.033] und enthält nicht die Null.

Die Verteilung dieser Daten ist symmetrisch und annähernd normalverteilt. Es wurde das Ziel erreicht.

Neue Familien der Potenz-Transformation

In der Realität sind selten alle betrachteten Originaldaten positiv. Aufgrund dessen mussten die Programme um Verfahren erweitert werden, welche mit negativen Daten umgehen können. Es sollen hier 3 mögliche Verfahren kurz erläutert werden.

Verschiebung der Daten

Wenn einige der Werte x_i negativ sind, kann eine positive Konstante ξ zu allen Beobachtungen hinzuaddiert werden, um sie positiv zu machen. Ersatzweise wird dann ξ in die Likelihoodfunktion mit eingeschlossen, $L_{max}(\xi, \lambda)$, um zu zeigen, das x_i ersetzt wurde mit $(x_i + \xi)$. Es muss nun die Maximum-Likelihood-Schätzung für ξ und λ gefunden werden (Box u Cox[1964]) [2].
Diese Variante, die verallgemeinerte Box-Cox-Transformation, wurde hier nur kurz vorgestellt und nicht implementiert.

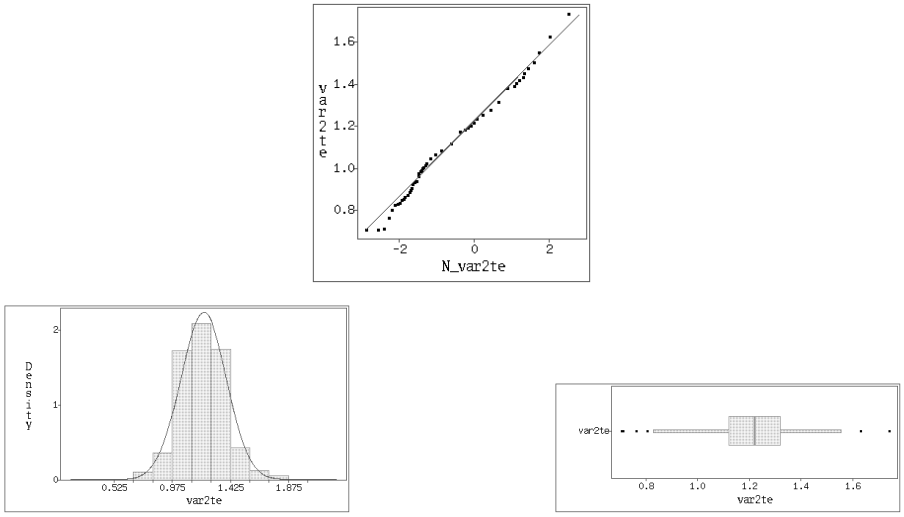


Abbildung 5: Daten der Variable 2 transformiert mit $\lambda = -0.07$

John und Draper

Untersuchungen von verschiedenen Transformations-Familien haben ergeben, dass die Box-Cox-Transformation häufig als „beste“ Wahl angesehen und oft in der Praxis benutzt wird. Sie macht eine schiefe Verteilung symmetrischer und, hoffentlich, normaler.
Die sogenannte Modulus-Transformation von John-Draper macht annähernd symmetrische Verteilungen normaler, durch die Verringerung der Krümmung (Kurtosis). Sie stellt damit eine Alternative für Fälle dar, in denen die Power-Transformation unpassend sein würde.
So dienen die Power- und Modulus-Transformationen etwas verschiedenen Zwecken.

Die Familie der monotonen Transformationen von John-Draper[1980] (stetig bei $\lambda = 0$) wird Modulus-Transformation genannt und für ziemlich symmetrische aber nichtnormale Fehlerverteilungen benötigt [4].

$$g_\lambda^J(x) = \begin{cases} \text{sign(x)} \left\{ \frac{(|x|+1)^\lambda - 1}{\lambda} \right\} & \text{für } \lambda \neq 0 \\ \text{sign(x)} \left\{ \log(|x|+1) \right\} & \text{für } \lambda = 0 \end{cases} \tag{6}$$

Sind alle Daten postiv, so sind die Modulus und die generalisierte Box-Cox-Transformation mit $\xi = 1$ äquivalent.

Der Maximum-Likelihood-Schätzer von λ kann wieder durch die Methode von Box-Cox erhalten werden. Die logarithmierte Likelihood-Funktion lautet

$$\log L_{max}^J(\lambda) \quad = \quad -\frac{n}{2} \log \left[\frac{1}{n} \sum\nolimits_{i=1}^{n} (g_\lambda^J(x_i) - \mu(\hat{\lambda}))^2 \right] + (\lambda - 1) \sum\nolimits_{i=1}^{n} \log \left(|x_i| + 1 \right) \tag{7}$$

Mit Hilfe der Lösungs-Methode nach Box-Cox kann die Formel (7), mit der Funktion (6), maximiert und dann gelöst werden.

Yeo und Johnson

Die durch John-Draper eingeführte Modulus-Transformation gilt für alle reellen Zahlen, aber beseitigt nur die Krümmung (Kurtosis). Eine neue Familie von Transformationen für schiefe Verteilungen mit negativen und positiven Werten ist deshalb gefragt.

Yeo-Johnson [5] stellen eine derartige Familie von Potenztransformation vor, die auf der ganzen reellen Zahlengeraden definiert sind und die für das Beheben der Schiefe und zur Erzielung ungefährer Normalverteilung angebracht sind. Ihre Eigenschaften, sind denen der Box-Cox-Transformation für positive Variablen ähnlich.

Um die Wahl der Yeo-Johnson-Transformationen zu motivieren, wird zuerst eine modifizierte Modulus-Transformation betrachtet, welche verschiedene Transformationsparameter für die positiven mit λ_+ und die negativen Zahlen mit λ_- einführt. Dann wird gefordert, dass die zweite Ableitung $\delta^2 g(\lambda_+, \lambda_-, x)/\delta x^2$ bei $x = 0$ stetig ist. Diese Bedingung stellt sicher, dass die Transformation glatt ist und impliziert $\lambda_+ + \lambda_- = 2$. Bei normalverteilten Originaldaten muss die Transformation sowohl bei den positiven als auch bei den negative Daten mit dem Wert 1 erfolgen, woraus ebenfalls folgt, dass $\lambda_+ + \lambda_- = 1+1 = 2$. Es wird die Yeo-Johnson-Transformation definiert mit:

$$g_\lambda^Y(x) = \begin{cases} \left\{(x+1)^\lambda - 1\right\}/\lambda & \text{für } x \geq 0, \lambda \neq 0 \\ \log(x+1) & \text{für } x \geq 0, \lambda = 0 \\ -\left\{(-x+1)^{2-\lambda} - 1\right\}/(2-\lambda) & \text{für } x < 0, \lambda \neq 2 \\ -\log(-x+1) & \text{für } x < 0, \lambda = 2 \end{cases} \tag{8}$$

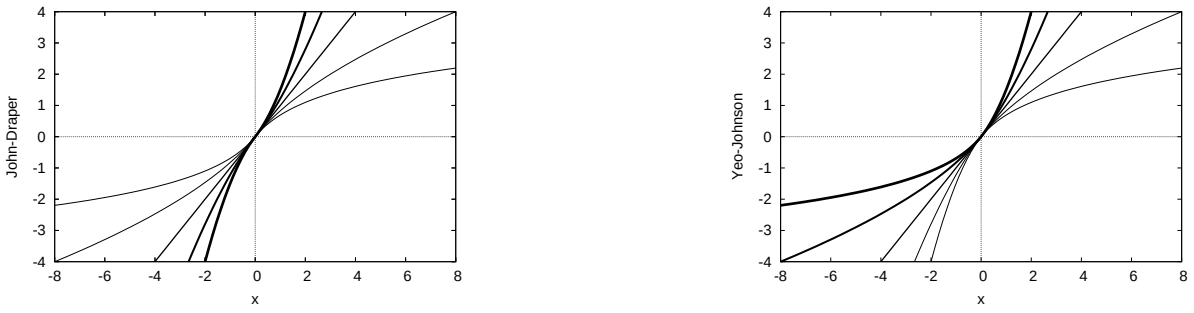


Abbildung 6: Ein Vergleich von (a) der John-Draper und (b) Yeo-Johson Transformation, mit $\lambda = 0, 0.5, 1, 1.5, 2$

Im positiven Bereich ist die Yeo-Johnson-Transformation mit der generalisierten Box-Cox-Transformation $\big((x+1)^\lambda - 1\big)/\lambda$ für $(x > 0)$ äquivalent, wo die Verschiebung mit der Konstanten 1 eingeschlossen ist. Ebenfalls erkennt man, wenn das Vorzeichen von x wechselt, dann ist der Wert von λ bei Yeo-Johnson ersetzt durch $(2 - \lambda)$ [5]. Bei John-Draper wird dagegen für alle Werte von x das λ benutzt.

Die Transformationsfunktion $g_\lambda^Y(x)$ hat folgende Eigenschaften:

1. $g_\lambda^Y(x) \geq 0$ für $x \geq 0$, und $g_\lambda^Y(x) < 0$ für $x < 0$;
2. konvex in x für $\lambda > 1$ und konkav in x für $\lambda < 1$;
3. eine stetige Funktion von (λ, x);
4. wenn $(g_\lambda^Y(x))^{(k)} = \delta^k g_\lambda^Y(x)/\delta\lambda^k$ dann gilt, für $k \geq 1, (g_\lambda^Y(x))^{(k)}$ ist stetig in (λ, x)

5. monoton wachsend in beiden Variablen λ und x;
6. konvex in λ für $x > 0$ und konkav in λ für $x < 0$;

Es wird angenommen, dass die transformierten Beobachtungen normalverteilt mit einem Mittelwert μ und einer Varianz σ^2 sind. Unter diesen Bedingungen ist die logarithmierte Likelihood-Funktion:

$$\log L^Y_{max}(\lambda) \quad = \quad -\frac{n}{2}\log\hat{\sigma}^2_Y(\lambda) + (\lambda-1)\sum\nolimits_{i=1}^{n} \text{sign}(x_i)\log\left(|x_i|+1\right) \tag{9}$$

Diese unterscheidet sich im wesentlichen von der logarithmierten Likelihoodfunktion der Box-Cox-Transformation (Gleichung (4)) durch das Vorzeichen in der Summe, sowie der unterschiedlichen Transformationsfunktion der Daten und deren Ableitungen, die in die Berechnung von $\hat{\sigma}^2$ und $(\hat{\sigma}^2)'$ mit eingehen. Sei $a = |x| + 1$.

$$\frac{\delta\ y(\lambda,x)}{\delta\ \lambda} = \begin{cases} \left(\frac{(a)^\lambda \log(a)\lambda - (a)^\lambda + 1}{\lambda^2}\right) & \text{für } x \geq 0, \lambda \neq 0 \\ \left(\frac{(a)^{2-\lambda}\log(a)(2-\lambda) - (a)^{2-\lambda}+1}{(2-\lambda)^2}\right) & \text{für } x < 0, \lambda \neq 2 \\ \left(\frac{\log^2(a)}{2}\right) & \text{für } (x \geq 0, \lambda = 0); (x < 0, \lambda = 2) \end{cases}$$

$$\frac{\delta^2\ y(\lambda,x)}{\delta\ \lambda^2} = \begin{cases} \left(\frac{\lambda^2(a)^\lambda(\log(a))^2 - 2\left(\lambda(a)^\lambda\log(a) - (a)^\lambda+1\right)}{\lambda^3}\right) & \text{für } x \geq 0, \lambda \neq 0 \\ \left(\frac{2(2-\lambda)(a)^{2-\lambda}\log(a) - 2(a)^{2-\lambda} + 2 - (2-\lambda)^2(a)^{2-\lambda}(\log(a))^2}{(2-\lambda)^3}\right) & \text{für } x < 0, \lambda \neq 2 \\ \left(\frac{\log^3(a)}{3}\right) & \text{für } (x \geq 0, \lambda = 0); (x < 0, \lambda = 2) \end{cases}$$

Auswertung

Nach der Berechnung des $\hat{\lambda}$ in beiden Verfahren, wird überprüft, welches λ eine bessere Anpassung der transformierten Daten an die Normalverteilung liefert. Dafür müssen die jeweiligen logarithmierten Likelihood-Funktionswerte, also Formel (7) und (9), berechnet werden. Das Maximum liefert die bessere Transformation für negative und positive Originaldaten.

Ausblick

Für die Bereinigung von nichtnormalen Verteilungen gibt es alternative Methoden, die auf dem Median anstelle des Mittelwertes basieren. Der Median ist dadurch charakterisiert, dass ebenso viele Werte drüber, wie drunter liegen. Solche Methoden sind beispielsweise die Robusten Verfahren.
Diese Methode kann noch bessere Ergebnisse liefern, da Ausreißer nicht mehr so einen großen Einfluss auf die Berechnungen haben.

Danksagung

Ich bedanke mich bei allen, die mich bei meiner Arbeit hier am Forschungszentrum tatkräftig unterstützt haben. Dazu gehören neben meinem Betreuer Dr. Wolfgang Meyer, dem der meiste Dank gebührt, auch Claudia Druska, Dr. Rüdiger Esser und dem NIC, welche das Gaststudenten-Programm 2005 ermöglicht haben.

Literatur

1. Box,G.E.P. und Cox,D.R. (1964), An Analysis of Transformation,
   Journal of the Royal Statistical Society, Series B, 26, 211-252
2. Seber, G.A.F., Transforming to Normality,
   Graphical and Data-Oriented Techniques
3. Draper, N.R. /Smith (1981), Transformation of the Response Variable
4. John,J.A. and Draper,N.R. (1980), An Alternative Family of Transformations,
   Applied Statistics, 29, 190-197
5. Yeo,In-Kwon and Johnson, R.A. (2000), An new family of power transformations to improve normality or symmetry,
   Biometrika Trust
6. Hoaglin, Emerson, John D., Mathematical Aspects of Transformation
7. Druska, Claudia (2004), Robuste statistische Verfahren für das Data Mining,
   Interner Bericht FZJ-ZAM-IB-2004-05, ZAM Forschungszentrum Jülich
8. Encyclopedia of Statistical Sciences

Evaluation of Hybrid Parallelization for Reconstruction of Large Phylogenies with MPI/OpenMP

Bui Quang Minh

Albert Ludwigs University of Freiburg, Institute for Computer Science

E-mail: minh@cs.uni-duesseldorf.de

Abstract:

Understanding phylogenetic relationship among species has been of tremendous interest since Darwin's invention of evolutionary theory (1859). Moreover, the recent boom in molecular sequence data has caused troubles with the efficiency of various phylogenetic reconstruction methods. Hence, a series of heuristics have been proposed trying to analyze the data in reasonable time. Recently, the IQPNNI algorithm [12], based on the maximum likelihood principle, has successfully combined several heuristics together. However, its running time is still not satisfactory.

Against this background, we propose a hybrid parallelization scheme for the IQPNNI algorithm using MPI/OpenMP, ensuring the scalability on hybrid parallel architectures, such as clusters of SMPs. Experiments on large datasets showed improved performance over a pure MPI parallelization [8]. Specifically, the hybrid parallelism scaled well as the number of processors increases, while the pure MPI presented a saturation effect. The runtime reduction and parallel scaling behavior suggest that hybrid parallelized IQPNNI is well suited to reconstruct large phylogenies.

Introduction

Motivation

Charles Darwin [4] said in his famous *evolutionary theory* that all species have evolved from a common ancestor under the pressure of *natural selection*. The structure of evolution can be formed as *evolutionary trees*. Such trees are also called *phylogenetic trees* or *phylogenies*. The interest in understanding this relationship among species has been there for 140 years since Darwin's foundation. Nevertheless, attempts to mathematically and computationally solve the problem have only been developed since the 1960s [6]. Among proposed frameworks, maximum likelihood has been observed to frequently achieve better results than others, despite its high computational expense.

Reconstructing phylogenies is proven to be a difficult problem [5; 2]. Additionally, the amount of available molecular sequence data has grown exponentially [GenBank; 1]. Thus, a series of heuristic methods have been proposed trying to infer phylogenetic trees in reasonable time. Nevertheless, the efficiency of those methods is still questionable in terms of both accuracy and speed.

Another promising consideration is parallel computing [11]. The popular parallel architecture nowadays is a cluster of Symmetric Multi Processors (SMPs). An SMP is a *shared memory* computer in which several processors have access to the same physical memory space. Such SMPs are clustered by a high bandwidth network to create a hybrid system. Processes on different SMP nodes can communicate with

each other using the *Message Passing Interface* [MPI; 10], an industry standard for programming on *distributed memory* systems. Inside an SMP, a process can be furthermore divided into several concurrent threads applying OpenMP, a standard for shared memory programming [3]. The SMP cluster motivates the application of hybrid programming models with both MPI and OpenMP in order to take full advantage of the hybrid parallel architecture. Care should be taken into account as such an approach does not always guarantee an improvement over the pure MPI parallelization [9].

IQPNNI algorithm

The IQPNNI algorithm [12] is a recently proposed heuristic to reconstruct phylogenetic trees successful in comparison to other well known methods. The IQPNNI consists of two major steps (Figure 1a). In the *initial step*, an initial tree is obtained using the *distance matrix*, the dissimilarities between pairs of sequences.

In the subsequent *optimization step*, the tree topology is reorganized to improve its quality. This is measured by the likelihood function, which will be discussed next. If the likelihood of the resulting tree exceeds that of the current best tree, then the current best tree is replaced by the new tree. The optimization step is repeated many times to thoroughly search the tree space. Iterations stop after a user-defined number of repetitions.

The sequential IQPNNI runs relatively slowly. Hence, a pure MPI parallelization pIQPNNI was made available [8]. pIQPNNI was shown to substantially reduce the running time, presenting a near optimal speedup up to 30 processors.

Contribution

The pure MPI parallel version pIQPNNI was furthermore ported to a hybrid parallelism with MPI/OpenMP. To this end, several time-consuming loops were rewritten so as to serve best for the OpenMP loop-level parallelism. We then evaluated its performance against the pure MPI parallelization on a cluster of SMPs. Details will be given in the following.

Parallelization

MPI parallelization of optimization step

The optimization step consumes 90% to 99% of the total running time, depending on the number of iterations and the size of the data. Hence, total running time will benefit greatly from its efficient parallelization. The parallelization was done using a master/worker scheme for distributed memory (Figure 1b): A master process is coordinating and collecting the results and workers are carrying out the actual computation.

Since the optimization step acts on the current best tree, iterations in the original version are not independent. To account for this, the sequential scheme was slightly modified: starting from the current best tree, every worker runs its own optimization step as explained before. After finishing one iteration, the worker always sends back its resulting tree to the master. The master receives and updates the current best tree if the received tree shows a higher likelihood. In such case, the master will broadcast the better tree to all other workers by non-blocking communication. This takes advantage of the time saved by possibly overlapping computation and communication [10]. The worker now starts its next iteration by first synchronizing its local tree with the master. The synchronization is not done during a single iteration at the workers. In addition, the master checks whether the stop condition applies and, if so, sends a stop message to all workers.

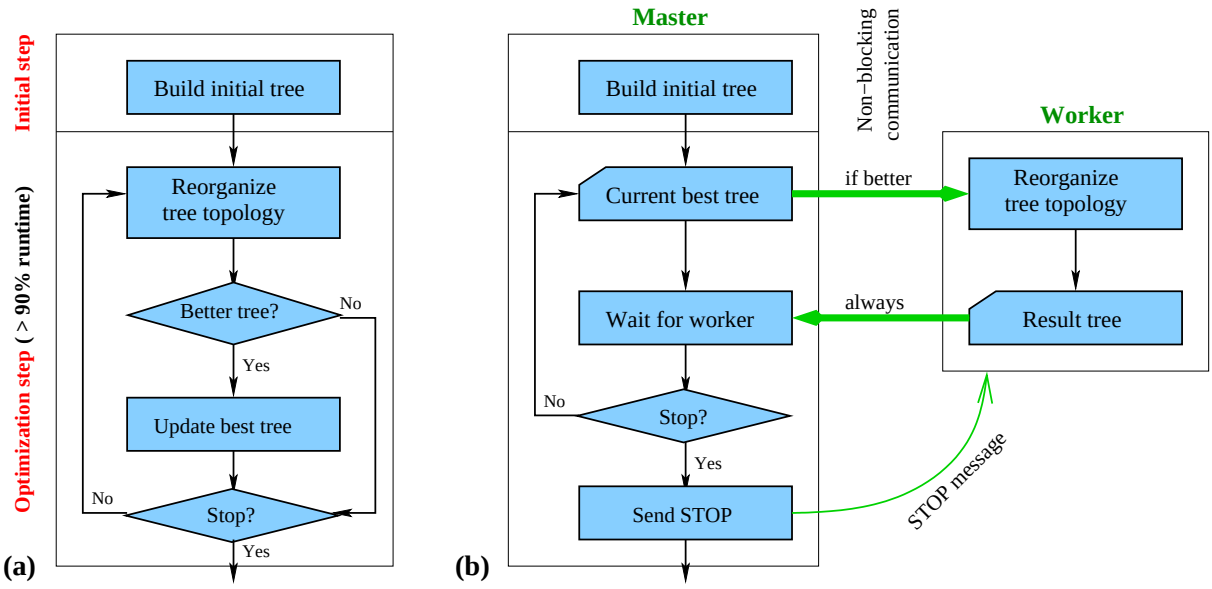


Figure 1: (a) Sequential and (b) MPI parallel scheme of IQPNNI algorithm.

OpenMP parallelization

IQPNNI dedicates most of its running time (at least 90%) only to calculate the likelihood of specific trees. This involves for-loops, whose iterations are independent of each other. Therefore, we focus on parallelizing these loops in the following way.

Given a dataset consisting of molecular sequences of n species. These sequences are aligned in a matrix of n rows (i.e. presenting species) and m columns, where m denotes the sequence length. In this alignment, columns are also called *sites*. In the probabilistic theory, the likelihood of the tree on these sequence data is measured by the conditional probability of the data given the tree [6]:

$$L(tree) = P(data|tree). \tag{1}$$

In the maximum likelihood framework, the likelihood works as an objective function, where we try to find a tree which maximizes its likelihood on the data. The computation of the likelihood can be very expensive. Hence, to simplify the approach, several assumptions have to be made. One of them is to assume that every site evolves independently of each other. According to this, the likelihood can now be rewritten as the product of the likelihood at each site:

$$P(data|tree) = \prod_{i=1}^{m} P(site_i|tree). \tag{2}$$

Normally, $P(site_i|tree)$ is very close to zero, and to dispose of numerical inaccuracies, one typically takes the logarithm of the likelihood function:

$$\log L(tree) = \sum_{i=1}^{m} \log P(site_i|tree). \tag{3}$$

Loop-level OpenMP parallelization can be easily employed by adding a pragma directive immediately before any “for” loop involving the computation of the likelihood as shown in listing 1. Caution was

Listing 1: Loop-level parallelism of computing likelihood.

```
#pragma omp parallel for private(logli_site) reduction(+: logli)
for (site=1; site <= nsites; site++) {
    logli_site = compute_log_likelihood(site);
    logli += logli_site;
}
```

taken into account by declaring some variables as *private*, to ensure thread safety when they are updated (e.g. logli_site in listing 1).

Hybrid parallelization of computing the distance matrix in the initial step

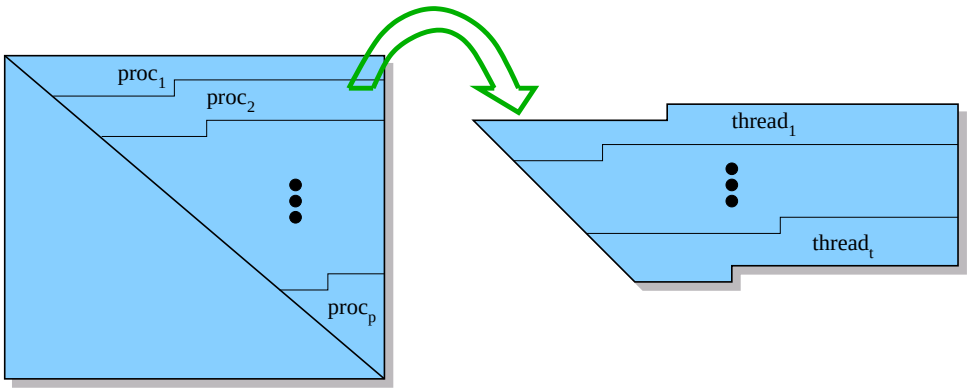


Figure 2: Hybrid MPI/OpenMP parallel scheme of computing distance matrix.

As seen before, the initial tree is obtained, based on the pairwise distance matrix. Each element of this matrix is independent of the others. Hence, a parallelization is straightforward. Since the distance matrix is symmetric, only the upper triangular part needs to be computed. The task assignment follows the *static chunking* method [7], i.e., if p denotes the number of processes and n is the number of sequences, the above diagonal part of the matrix will be divided into p approximately equal contiguous regions. Each process is responsible for computing $n(n-1)/2p$ pairwise distances belonging to its assigned region (Figure 2).

The same strategy is applied to share work among threads in each process. If t is the number of threads per process, then the assigned region of a process is again partitioned into t parts for t threads. Once finished, the results are shared among all processes with an MPI_Allgather call.

Performance Study

System configuration

We tested the program on the JUMP (Juelich Multi Processor) system, a cluster of 41 IBM Regatta p690+ SMP nodes. Each node has 32 Power4+ processors of 1.7 GHz. We used up to 128 CPUs on this supercomputer to test the program. We compiled the program under the MPI-supported IBM C++ compiler (mpCC_r) with the following flags: `-qsmp=omp -O3 -q64`. The option `-qsmp=omp` enables the OpenMP support and `-q64` produces 64-bit executable code. Although the option `-O3` without the flag `-qstrict` on the IBM xl compiler family is allowed to alter program semantics, control runs against an

executable compiled with `-O2` retained the same results. The executable compiled with `-O3`, however, showed improved overall performance.

Datasets

Table 1: The datasets used for analysis and sequential runtime.

Type	Name	#Seqs	#Sites	#Iterations	Initial step	Opt. step	Total
DNA	218dna[1]	218	4182	150	70s	47m:55s	49m:05s
AA	74aa[2]	74	4013	50	77s	32m:41s	33m:58s

[1] prokaryotic sequences from the small ribosomal subunit.

[2] vertebrate amino acid sequences.

The experiments were conducted on two real datasets (Table 1). The well known dataset 218dna, also called *ssu rRNA*, can be downloaded from the Ribosomal Database Project II http://rdp.cme.msu.edu/. The unpublished protein dataset 74aa was kindly provided by colleagues H.A. Schmidt and D. Liebers. Both datasets were previously analyzed by Minh *et al.* [8], which showed a very good scaling of the pure MPI parallelization up to 30 CPUs. Table 1 displays the sequential running time. The initial step took only about 3% of the whole time on both datasets.

Pure OpenMP

Firstly, we measured the performance of the pure OpenMP parallelization on a JUMP node using up to 32 processors as depicted in figure 3. Figure 3a shows the speedup for the computation of the distance matrix during the initial step. Interestingly, the speedup on both datasets is optimal up to 8 processors and suddenly remains at this level when we increased the number of threads to 16 and 32. This behavior is even worse for the total running time (Figure 3b): it also scaled well until 8 threads and then the speedup drops sharply with more than 8 threads. The runtimes with 16 threads on dataset 218dna and with 32 threads on both datasets are even greater than the sequential time, and thus ignored in the figure.

This break-in in overall performance seems to be connected to compiler issues, local to the JUMP system.

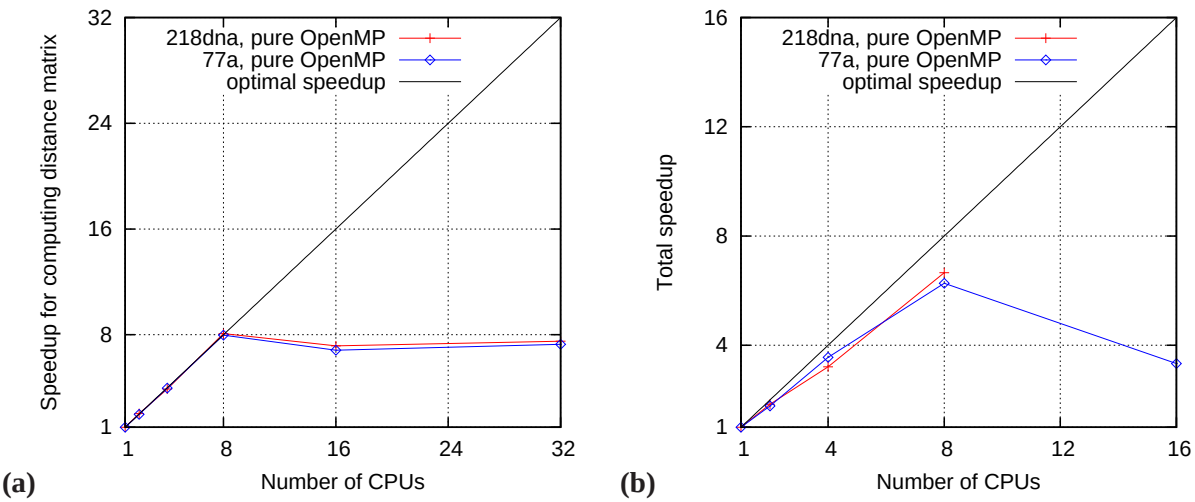


Figure 3: Pure OpenMP speedup for (a) computing distance matrix and (b) the whole program.

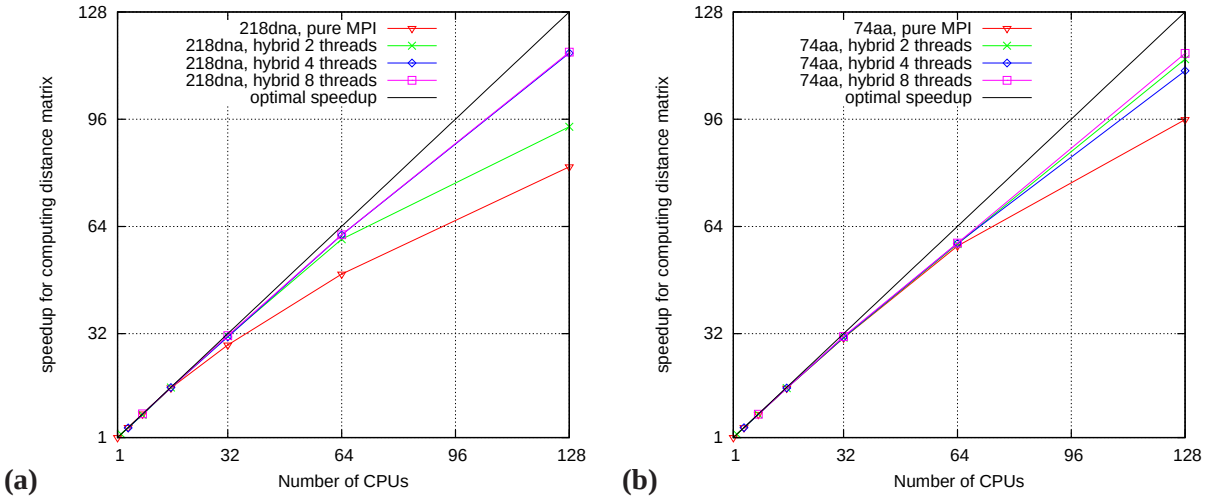


Figure 4: Speedup of computing distance matrix for dataset (a) 218dna and (b) 74aa.

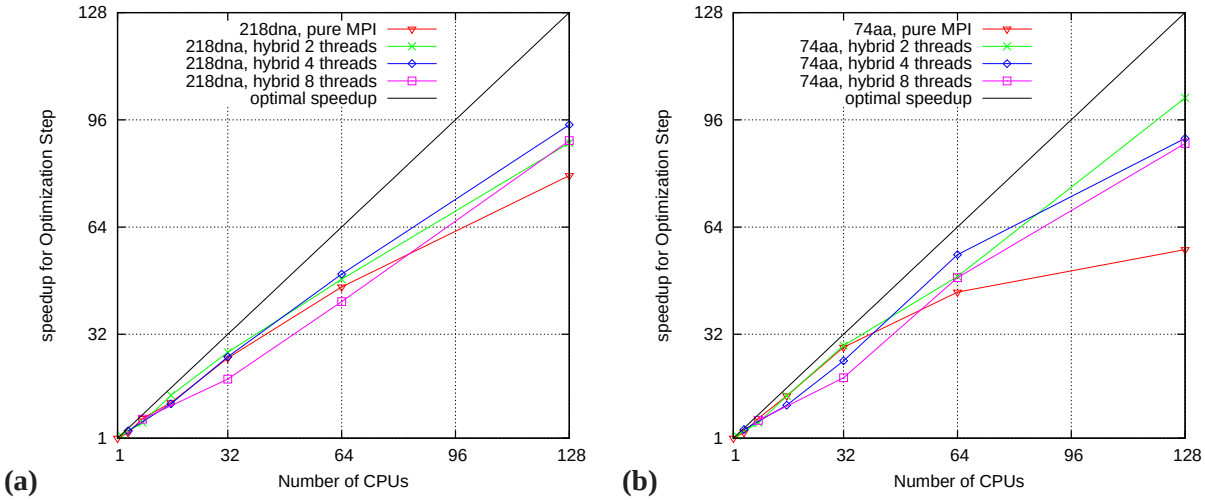


Figure 5: Speedup of optimization step for dataset (a) 218dna and (b) 74aa.

Due to the limited time and an expected imbalance with too many threads in the hybrid version, this phenomenon was not further investigated. For that reason, we restricted to a maximum of 8 threads per process in the subsequent analysis.

Pure MPI vs. hybrid MPI/OpenMP

We tested the performance of the pure MPI parallelization against the newly implemented hybrid version with 2, 4 and 8 OpenMP threads per process. Figure 4 displays the speedup in computing the distance matrix. On the dataset 218dna, the pure MPI and the 2-threads hybrid version shows a saturation effect with more than 32 processes, i.e. from 32 and 64 CPUs, respectively. On the contrary, the speedup with 4- and 8-threads is near optimal. This results from smaller amounts of communication, as the number of processes decreases when the number of threads per process increases. On the 74aa dataset, the scalability is better than on the 218dna dataset and we only observe the saturation on the pure MPI version. This is due to the fact that the distance matrix for the 218dna dataset is larger than that for the 74aa. As a consequence, the communication overhead on the 218dna dataset is larger, thus reducing its performance.

Figure 5 shows the speedup for the optimization step. On both datasets, we see that the pure MPI ver-

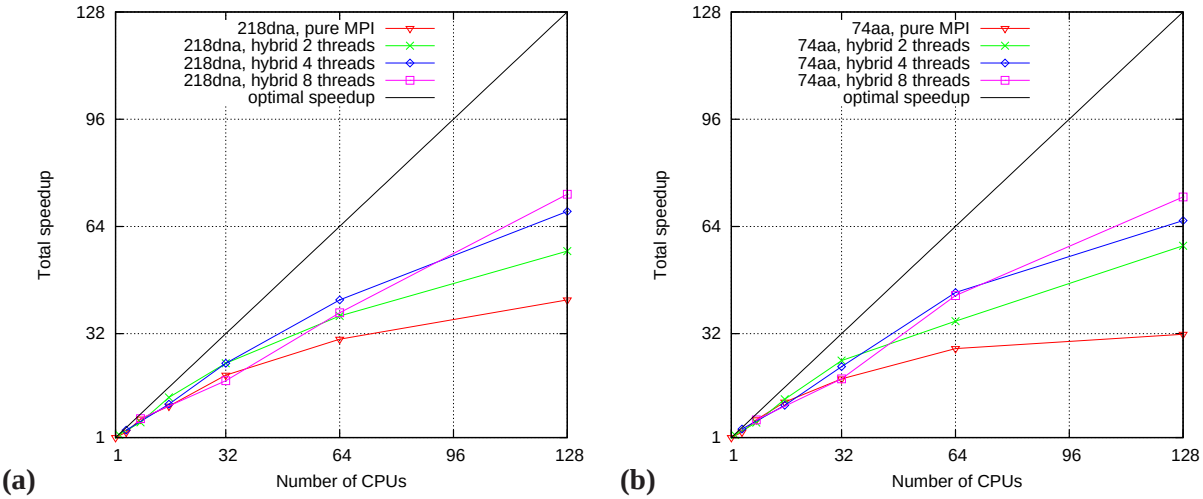


Figure 6: Total speedup for dataset (a) 218dna and (b) 74aa.

Table 2: Runtime on dataset 218dna with 128 CPUs.

#Processes	#Threads	Initial step	Opt. step	Total	Speedup
128	pure MPI	34s	36s	70s	42.1
64	2	20s	32s	52s	56.6
32	4	13s	30s	43s	68.5
16	8	08s	32s	40s	73.6

sion goes to saturation with more than 64 processors. In contrast, the 2-threads hybrid version always outperforms the pure MPI with a near linear speedup. The 4- and 8-threads parallelization present poor speedup with less than 8 processes (i.e. 32 and 64 CPUs, respectively). However, this effect disappears with 16 processes or more and the scaling is comparable to that with 2 threads. This can be caused by the master process consuming actually only one processor and the other $t-1$ processors are unused. So a fraction of $\frac{t-1}{pt}$ CPUs are wasted during the optimization step. The effect will be large if p is small and otherwise become lower if p increases.

In addition, we observe that the speedup of the pure MPI version on the dataset 74aa is quite poor when running on 128 CPUs. This is, however, due to the fact that we only set the number of iterations to 50 on this dataset. If each iteration on each worker process consumed roughly the same amount of time, then we would need only 50 workers to finish the whole optimization step. That means the other 77 workers would be omitted. This affect will not occur if we increase the number of iterations to at least the number of MPI processes.

The speedup of the whole program is depicted in figure 6. The hybrid run with 8-threads scales best on 128 processors and is comparable to those with 4- and 2-threads on fewer CPUs. This shows a tendency that raising the number of threads per process will improve the performance with the growing number of CPUs.

Moreover, the speedup curves are not similar to those for the optimization step since the main part of the initial step is carried out sequentially in the MPI parallelism. As a result, its runtime proportion will be more significant with the increasing number of CPUs. Table 2 displays the the running time with 128 CPUs on the dataset 218dna. The initial step of the pure MPI run consumed 50% of the total time and the time reduction for the hybrid parallelization is mainly due to the shorter time used in the initial step.

Conclusions and Future Work

In this study, we applied two strategies to port a pure MPI parallelization of the IQPNNI algorithm into a hybrid MPI/OpenMP parallelism. Firstly, the data partition scheme by the static chunking method for the MPI processes is also employed for the OpenMP threads. It is implemented for the computation of the distance matrix and shown to be very efficient. Secondly, the loop-level parallelism is applied to the time-consuming for-loops calculating the likelihood of the phylogenies. These loops appeared at a low level and all MPI functions are called outside the parallel OpenMP regions. Hence, this ensures portability even when the MPI libraries are not thread safe.

Analyzes on two real datasets showed improved performance of the hybrid parallelization over the pure MPI on a cluster of SMPs. We tested up to 128 CPUs and 8 threads per process. With large numbers of processors, the pure MPI implementation indicated a saturation effect. In contrast, the hybrid version scaled better, especially when increasing the number of threads. This is due to the fact that we make full use of the capabilities of the hybrid architecture.

The current research also opens a perspective in which the program can still be enhanced. The master process actually consumes only one CPU during the optimization step, i.e., the remaining $t-1$ processors are unused, where t is the number of threads per process. Future direction would be to group these idle processors into an additional worker using a nested OpenMP parallel region, which is not supported by all OpenMP implementations.

Acknowledgements

I would like to thank Marc-AndrÃ© Hermanns for his continual advice during the guest student programme 2005 at the Research Center Juelich. The kind support in organizational matters by Ruediger Esser is greatly appreciated. The thanks also goes to Brian Wylie for his careful reading of the manuscript. The use of computing facilities at ZAM/NIC is gratefully acknowledged.

References

1. Benson, D. A., Karsch-Mizrachi, I., Lipman, D. J., Ostell, J. and Wheeler, D. L. (2005) GenBank. *Nucl. Acids Res.*, **33**, D34–D38.
2. Chor, B. and Tuller, T. (2005) Maximum likelihood of evolutionary trees is hard. In *Proceedings of the 9th Annual International Conference on Research in Computational Molecular Biology (RECOMB 2005)*, volume 3500 of *Lecture Notes in Computer Science*, pp. 296–310, ACM Press, New York, USA.
3. Dagum, L. and Menon, R. (1998) OpenMP: An industry-standard API for shared-memory programming. *IEEE Comput. Sci. Eng.*, **5**, 46–55.
4. Darwin, C. (1859) *On the Origin of Species by Means of Natural Selection*. John Murray, London.
5. Felsenstein, J. (1978) The number of evolutionary trees. *Syst. Zool.*, **27**, 27–33.
6. Felsenstein, J. (2004) *Infering Phylogenies*. Sinauer Associates, Sunderland, Massachusetts.
7. Hagerup, T. (1997) Allocating independent tasks to parallel processors: An experimental study. *J. Parallel Distrib. Comput.*, **47**, 185–197.
8. Minh, B. Q., Vinh, L. S., von Haeseler, A. and Schmidt, H. A. (2005) pIQPNNI-parallel reconstruction of large maximum likelihood phylogenies. *Bioinformatics*.
9. Smith, L. and Bull, M. (2001) Development of mixed mode MPI/OpenMP applications. *Scientific Programming*, **9**, 83–98.
10. Snir, M., Otto, S. W., Huss-Lederman, S., Walker, D. W. and Dongarra, J. (1998) *MPI: The Complete Reference - The MPI Core*, volume 1. 2nd edition, The MIT Press, Cambridge, Massachusetts.
11. Trelles, O. (2001) On the parallelisation of bioinformatics applications. *Brief. Bioinform.*, **2**, 181–194.
12. Vinh, L. S. and von Haeseler, A. (2004) IQPNNI: Moving fast through tree space and stopping in time. *Mol. Biol. Evol.*, **21**, 1565–1571.

Automatic Performance Analysis of Parallel Programs with KOJAK

Jens Doleschal

Department of Mathematics
Institute of Scientific Computing
Center for Information Services and High Performance Computing
Dresden University of Technology

E-mail: doleschal@zhr.tu-dresden.de

Abstract: A reason why performance analysis of parallel programs is a difficult task is that users frequently cannot predict the performance of their application because of the very large and complex tracing data set produced by conventional parallel performance tools. As a consequence, automatic analysis tools are required to identify frequently occurring and well-defined performance problems automatically. The automatic performance analysis toolset KOJAK provides a complete tracing-based solution for MPI, OpenMP or hybrid parallel applications. To evaluate its usability and effectiveness for larger applications, two complex ASCI benchmarks (UMT2K and SMG2000) and a very large community chemistry application (CP2K) have been instrumented and analyzed with KOJAK. During instrumentation of an executable with KOJAK several common problems can appear. This report describes them and gives solution hints. For securing that KOJAK runs correctly with Fortran MPI applications on different systems, the KOJAK C MPI testsuite has been extended to Fortran.

Introduction

Often parallel applications are running inefficiently with a lot of bottlenecks. These bottlenecks lead to a degradation of the performance of the application in general and of its parallel speedup. The goal of performance analysis tools is to help to determine performance problems and to validate tuning decisions. There are two basic strategies: profiling and tracing.

Profiling is the recording of summary information, e.g., execution time or number of calls, of program entities like subroutines or basic blocks. It provides a quick, low cost overview, but often fails to help exposing performance problems because not enough performance data is provided.

Tracing is the recording of information about significant events during execution of the program and it can be used to reconstruct the dynamic behavior of an application. The information is saved in event records, which usually contain a timestamp, CPU ID, thread ID, event type and other event specific information. An event trace is a sequence of event records sorted by time. So while tracing provides a lot of detailed information which allows to (potentially) find performance bottlenecks, this is also its main problem. A manual analysis of the large and complex tracing data set is a very difficult task, so a typical programmer fails to use this technique effectively. To solve this problem, the automatic performance analysis toolset KOJAK was developed. KOJAK instruments parallel applications, collects the tracing data, scans the trace file for event patterns that indicate a performance problem and presents the problems

found.

In this work, the usability of KOJAK for large application programs is analyzed. In the next section, KOJAK is presented in more detailed. Then I describe the parallel applications used in this study, how difficult is was to use KOJAK to analyze their performance and present some results obtained with KOJAK. In addition, I report on my work on enhancing the KOJAK test suite with Fortran modules.

KOJAK

KOJAK is the "Kit for Objective Judgment and Automatic Knowledge-based detection of bottlenecks". It was designed at the Research Center Juelich and the Innovative Computing Laboratory at the University of Tennessee. KOJAK instruments C, C++ and Fortran parallel applications based on MPI, OpenMP, or a combination of the two. Figure 1 gives an overview of KOJAK's architecture and its components.

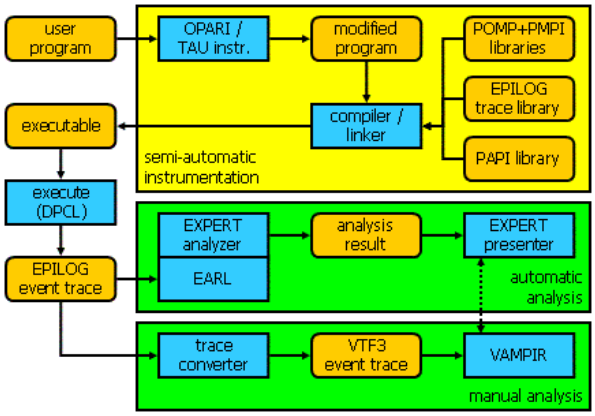


Figure 1: KOJAK architecture

The KOJAK analysis process consists of two parts: a semi-automatic multi-level instrumentation of the user application followed by an automatic analysis of the generated performance data. The first subprocess is semi-automatic because it requires user interaction to modify the Makefile, maybe to instrument user functions, and execute the application manually. Specific commands that can be put in front of the actual compile and link commands automatically call the required instrumentation tools, like OPARI [3], and link to all necessary run-time libraries.

Automatic instrumentation of user functions could be done using hidden, undocumented compiler switches. The problems of this approach are that it instruments every function, so that the trace file may become too large, and it doesn't work on every machine. So it is recommended to profile the application first, determine the significant routines and instrument them manually with `POMP INST` directives. Then the application's source code is supplied to OPARI, which processes these POMP directives but also performs automatic instrumentation of OpenMP constructs. Instrumentation for MPI is accomplished with a PMPI interposition library, which generates MPI-specific events by intercepting calls to MPI functions. All MPI, OpenMP, and user-function instrumentation call the EPILOG [1] runtime library, which provides mechanisms for buffering and tracefile creation. Also, if the user wants to record hardware counters as part of the trace file, the application needs to be linked to the PAPI [4] library. At the end of the instrumentation process the user has a fully instrumented executable.

Running this executable generates a tracefile in the EPILOG format. After program termination, the tracefile is fed into the EXPERT analyzer. The analyzer generates an analysis report, which serves as input for the EXPERT presenter, called CUBE. In addition it is possible to convert EPILOG traces to VTF3 format and analyze them manually with the VAMPIR [7] event trace analysis tool.

For users who need a cross-experiment analysis, KOJAK includes a performance data algebra, that allows the execution of different arithmetic operations on CUBE instances. These operations are difference, merge, and mean. The result can be displayed with CUBE like a normal experiment.

Benchmarks

General

The main goal of my work with KOJAK was to test its usability for larger applications. Therefore, three different programs have been instrumented and analyzed.

For the experiments, the JUMP system of Research Centre Juelich was used which is a 41-node IBM system with 32 Power4 CPUs in each node. For comparing different measurements, it is useful to test a set of typical configurations that are comparable, e.g., to compare an MPI run on 1 node with a hybrid run on 1 node. Therefore the following configurations have been tested.

	MPI ranks $\times$ OpenMP Threads
1 Node	32×1
	4×8
4 Nodes	128×1
	16×8

Table 1: Configurations

During the experiment, the following two basic problems occurred:

If the source code of an application is distributed across several directories the automatic instrumentation with kinst-pomp will fail, because OPARI will create separate OPARI helper files in each source directory instead of the unique ones necessary. Therefore a manual workaround is necessary. First, one has to use `kinst-pomp -rcfile $(RCFILE) --` instead of the usual `kinst-pomp` to instrument the source files of the application. In addition, before linking the executable, in the directory where the executable is build, the user has to use the following command to get unique OPARI helper files: `opari -rcfile $(RCFILE) -table $(TABFILE)`, where RCFILE is the absolute path of the OPARI rcfile and TABFILE the absolute path of the OPARI tabfile. Finally, the user has to create symbolic links to the local OPARI instrumentation files in each directory with the command `ln -s <path-to-src-dirs>/*.opari.inc`.

Second, because every JUMP node has its own system clock and the latency of the MPI messages is very small, there are also problems with KOJAK's time synchronization on JUMP. So the user has to correct the trace after the measurement with the `elg_timecorrect` tool. The only disadvantage is, that the tool only works correctly with pure MPI applications. With OpenMP or hybrid applications running on more than one node the correction fails. Therefore it is not possible at the moment to create useful traces of applications using OpenMP and running on more than one node.

In the following, I describe the results for each of the three applications.

UMT2K

General

The UMT benchmark is a 3D, deterministic, multigroup, photon transport code for unstructured meshes. It consists of 136 files in 11 directories with a total of 20,985 lines of code. It also uses two external tools with additional 228 files and 160,738 lines of code.

The code works on unstructured meshes, which is generated at runtime using a two-dimensional unstructured mesh (input) and extruding it in the third dimension a user-specified amount. This allows the generation of a wide variety of input problem sizes and facilitates "constant work" scaling studies. The MPI-based parallelism in the Fortran portion uses mesh decomposition to distribute the mesh across the specified MPI tasks. The OpenMP-based parallelism in the C kernel is from a single OpenMP directive in the kernel snflwxyz.c and is suited for utilizing shared-memory multiprocessor nodes. The directive divides the ordinates among the OpenMP threads.

The kernel computation is written in C, but was originally written in Fortran90 and translated to C by using a f2c converter. All array accesses are defined in the `array_wrappers.h` to preserve the "column major" access pattern of Fortran.
The computation kernel consists of 20 files with a total amount of 2831 lines and it typically dominates the execution time of the benchmark.

Building the Code

1. Download umt1.2.1.tar and untar the file with `tar xvf umt1.2.1.tar`.

2. Change into directory "umt2k/metis_src" and untar metis-4.0.tar with `tar xvf metis-4.0`.

3. Change into directory "metis-4.0" and edit the "Makefile.in": set the C compiler and optionally the compiler flags. The following options can be used on JUMP.

   ```
   CC = mpcc_r
   OPTFLAGS = -O2
   COPTIONS = -q64
   LDOPTIONS = -q64
   AR = ar -rv -X 32_64
   RANLIB = ranlib
   ```

4. Install Metis with `make`.

5. Change into directory "umt2k/silo_src" and unpack the silo001023.sh with `sh silo001023.sh`.

6. Change into directory "silo001023" and follow the install-instructions in the file "INSTALL_NOTES". If the code must be build for the first time, start to read at line 94, otherwise start at line 1.
   On IBM RS 6000 (AIX) the following configuration is required.
   ```
   env CFLAGS="-O2 -q64" AR="ar -rv -X 32_64"
   ./configure --enable-parallel
   ```

7. Install Silo with `make`.

8. Change into directory into "umt2k/umt2k_src/include" and determine which `.mk` is being used for your platform.

9. Edit the `*.mk` files. Add `-q64` to the compiler flags and change the C preprocessor flag to `F` for the fixed form and to `F90` for the free form.
   The following changes were added in the file aix_2.mk for JUMP.

```
# OPARI, KOJAK definitions
RCFILE     := /home3/zdvex/gstu0503/benchmark/umt2k/umt2k_src/ \
opari.rc
TABFILE    := /home3/zdvex/gstu0503/benchmark/umt2k/umt2k_src/ \
opari.tab.c
KINST-POMP := kinst-pomp -rcfile $(RCFILE) --
OTABGEN    := opari -rcfile $(RCFILE) -table $(TABFILE)
OTALLN     := ln -s */*.opari.inc .
RmOpari    := *.mod.* *opari.* */*.mod.* */*opari.*

# Fortran90 compiler definitions
SerialF90     := $(KINST-POMP) xlf90
PThreadF90    := $(KINST-POMP) xlf90_r
MPIF90        := $(KINST-POMP) mpxlf90_r
MPIPThreadF90 := $(KINST-POMP) mpxlf90_r

# C compiler definitions
SerialCC     := $(KINST-POMP) xlc
PThreadCC    := $(KINST-POMP) xlc_r
MPICC        := $(KINST-POMP) mpcc_r
MPIPThreadCC := $(KINST-POMP) mpcc_r
```

10. Edit the `Makefile` in directory "umt2k_src" and add the following code at line 350.

```
@echo ""
@echo "CREATING OPARI.TAB.INC"
$(OTABGEN)
@true
$(OTALLN)
@true
@echo "FINISHED CREATING OPARI.TAB.INC"
@echo ""
```

    In addition, add a new line with `$(rm) $(RmOpari)` after line 312.

11. Instrument the main and other significant routines with `POMP` directives.

```
control/radtr.f:136:      !POMP$ INST BEGIN(radtr)
control/radtr.f:391:      !POMP$ INST END(radtr)

drive/cntrlrad.f:176:      !POMP$ INST BEGIN(cntrlrad)
drive/cntrlrad.f:654:      !POMP$ INST END(cntrlrad)
drive/main.f:210:      !POMP$ INST INIT
drive/main.f:211:      !POMP$ INST BEGIN(umt2k)
drive/main.f:979:      !POMP$ INST END(umt2k)

rt/rswpmd.f:98:      !POMP$ INST BEGIN(rswpmd)
```

```
rt/rswpmd.f:181:      !POMP$ INST END(rswpmd)
rt/rtmainsn.f:324:      !POMP$ INST BEGIN(rtmainsn)
rt/rtmainsn.f:800:      !POMP$ INST END(rtmainsn)
rt/rtorder.f:41:      !POMP$ INST BEGIN(rtorder)
rt/rtorder.f:56:      !POMP$ INST END(rtorder)

snac/snflwxyz.c:155:    #pragma pomp inst begin(snflwxyz)
snac/snflwxyz.c:536:    #pragma pomp inst end(snflwxyz)
snac/snmmnt.c:61:    #pragma pomp inst begin(snmmnt)
snac/snmmnt.c:104:    #pragma pomp inst end(snmmnt)
snac/snqq.c:72:    #pragma pomp inst begin(snqq)
snac/snqq.c:249:    #pragma pomp inst end(snqq)
snac/snswp3d.c:116:    #pragma pomp inst begin(snswp3d)
snac/snswp3d.c:565:    #pragma pomp inst end(snswp3d)
snac/snxyzref.c:64:    #pragma pomp inst begin(snxyzref)
snac/snxyzref.c:157:    #pragma pomp inst end(snxyzref)
```

12. Run the commands `gmake distclean` and `gmake optimize`.
13. To run the benchmark, change into directory "TEST/smalldir", modify line 31 in file rtin to the number of MPI processes and run the executable with e.g.,
`llrun -p 4 -t 1 ../../umt2k -procs n` on JUMP.

Create a new problem

To generate a new problem, create a new subdirectory and copy smrtin and opacfile from the "TEST" directory and one of the rtin files supplied with the UMT distribution into the new directory. To use multiple MPI processes, change the number of domains on line 31 in file rtin to the number of MPI processes used. There are several ways to influence the execution time. The execution time for a problem normally scales like the product of the number of zones, the number of directions, the number of groups, and the number of time steps.
To change the number of time steps, change tmax on line 121. To change the number of zones in the problem, multiply the numbers in the final column of lines 48 through 56 by the same factor. The number of groups can be changed to an arbitrary value on line 23. Additionally, add or remove a corresponding number of lines at the end of the Frequency Group Definition beginning on line 94.

Results

By comparing the runs 32×1 and 128×1 and using the performance property view (Figure 2, left) it is easy to see that the program doesn't scale with the number of processes. More MPI processes need especially more time for MPI_Init, for communication and for synchronization. By using the call tree (Figure 2, middle), one can quickly locate the call paths of the identified performance problems. Here the major source for the synchronization problem is the MPI_Barrier. With the system tree (Figure 2, right) it is easy to see how the problem is distributed across the machine.
All in all the program needs more time for running the additional MPI expenditure than it saves by distributing the work of the execution phase.

Introduction of OpenMP multithreading (Figure 3) increases (as expected) the time of idle threads and the OpenMP time. The time for running MPI is reduced by increasing the number of threads. However, use of OpenMP leads to an increase of the overall execution time.

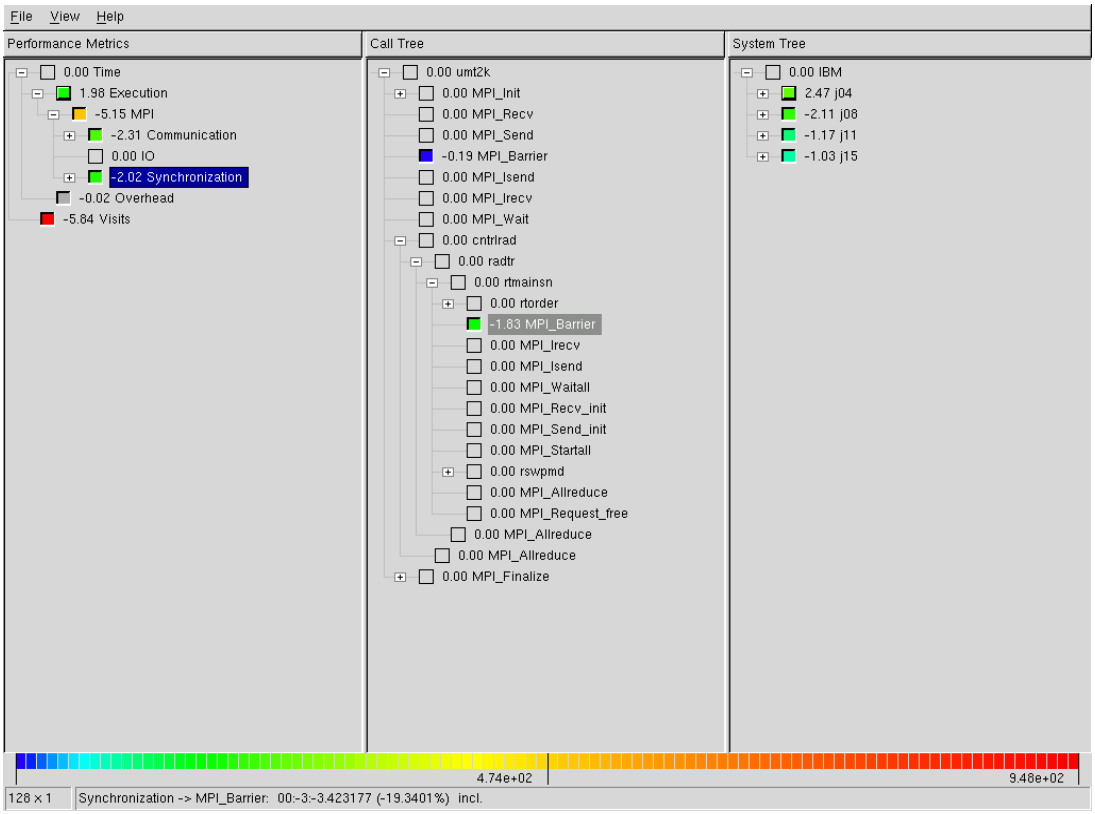


Figure 2: UMT2K Diff 32×1 vs. 128×1

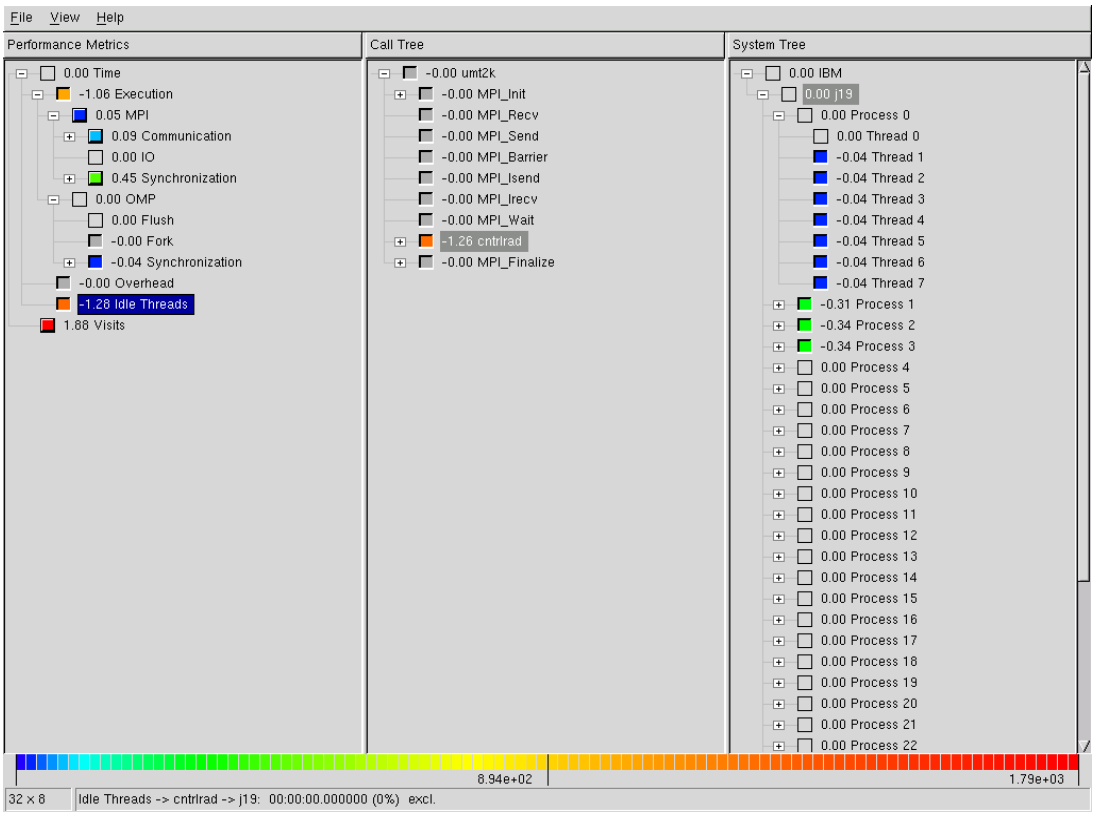


Figure 3: UMT2K Diff 32×1 vs. 4×8

SMG2000

General

SMG2000 is a parallel semi-coarsening multigrid solver for the linear systems arising from finite difference, finite volume or finite element discretizations of a diffusions equation on logically rectangular grids. It consists of 70 files in 5 directories with approximately 28,000 lines.

Building the Code

1. Download smg2000.tar and untar the file.
2. Change into directory "smg2000".
3. Edit the Makefile.include. The following configuration was used.

```
# OPARI, KOJAK definitions
RCFILE     = /home3/zdvex/gstu0503/benchmark/smg2000/test/opari.rc
TABFILE    = /home3/zdvex/gstu0503/benchmark/smg2000/test/opari.tab.c
KINST-POMP = kinst-pomp -@ -rcfile ${RCFILE} --
OTABGEN    = opari -rcfile ${RCFILE} -table ${TABFILE}
OTALLN     = ln -s */*.opari.inc
RmOpari    = *.mod.* *opari.* */*.mod.* */*opari.*

# set the compiler here
CC = ${KINST-POMP} mpcc_r

# set compile flags here
INCLUDE_CFLAGS = -q64 -O -qsmp=omp -DHYPRE_USING_OPENMP -DTIMER_USE_MPI

# set link flags here
INCLUDE_LFLAGS = -lm -q64
```

4. Edit the following files.
   - `Makefile` in directory smg2000 with `@echo "RmOpari" @rm -f ${RmOpari}` at line 45 and 56.
   - `test/Makefile` with

     ```
     @echo ""
     @echo "CREATING OPARI.TAB.INC"
     ${OTABGEN}
     @true
     ${OTALLN}
     @true
     @echo "FINISHED CREATING OPARI.TAB.INC"
     @echo ""
     ```

     at line 54
5. Instrument the main and other significant routines with `POMP` directives.

```
in directory struct_ls:

cyclic_reduction.c:814:   #pragma pomp inst begin(hypre_CyclicReduction)
cyclic_reduction.c:1148:   #pragma pomp inst end(hypre_CyclicReduction)
semi_interp.c:176:   #pragma pomp inst begin(hypre_SemiInterp)
semi_interp.c:311:   #pragma pomp inst end(hypre_SemiInterp)
semi_restrict.c:174:   #pragma pomp inst begin(hypre_SemiRestrict)
semi_restrict.c:279:   #pragma pomp inst end(hypre_SemiRestrict)
smg.c:398:#pragma pomp inst begin(hypre_SMGSetStructVectorConstantValues)
smg.c:422:#pragma pomp inst end(hypre_SMGSetStructVectorConstantValues)
smg3_setup_rap.c:287:   #pragma pomp inst begin(hypre_SMG3BuildRAPSym)
smg3_setup_rap.c:933:   #pragma pomp inst end(hypre_SMG3BuildRAPSym)
smg_axpy.c:47:   #pragma pomp inst begin(hypre_SMGAxpy)
smg_axpy.c:77:   #pragma pomp inst end(hypre_SMGAxpy)
smg_residual.c:192:   #pragma pomp inst begin(hypre_SMGResidual)
smg_residual.c:306:   #pragma pomp inst end(hypre_SMGResidual)

struct_mv/struct_axpy.c:46:   #pragma pomp inst begin(hypre_StructAxpy)
struct_mv/struct_axpy.c:76:   #pragma pomp inst end(hypre_StructAxpy)

test/smg2000.c:69:   #pragma pomp inst init
test/smg2000.c:70:   #pragma pomp inst begin(main)
test/smg2000.c:640:   #pragma pomp inst end(main)
```

6. Install with `make`. This will produce an executable file "smg2000" in the "test" directory.
   Other available targets are:
   `make clean` (delete object files)
   `make veryclean` (delete object files, libraries and executables)

7. To run the benchmark change into directory "test", and run the executable with e.g.,
   `llrun -p 4 -t 1 ./smg2000 -n 10 10 10 -c 0.1 1.0 10.0 -P 2 2 1` on JUMP.

Create a new problem

A new problem can be generated by changing the commandline options. All of the arguments are optional. The following options are available:

- `-n <nx> <ny> <nz>` : problem size per block
- `-P <Px> <Py> <Pz>` : processor topology
- `-b <bx> <by> <bz>` : blocking per processor
- `-c <cx> <cy> <cz>` : diffusion coefficients
- `-v <n_pre> <n_post>` : number of pre and post relaxations
- `-d <dim>` : problem dimension (2 or 3)
- `-solver <ID>` : solver ID (default = 0)
  - 0 - SMG
  - 1 - CG with SMG precond

– 2 - CG with diagonal scaling
– 3 - CG

The most important options are the `-n` and the `-P` options.

Results

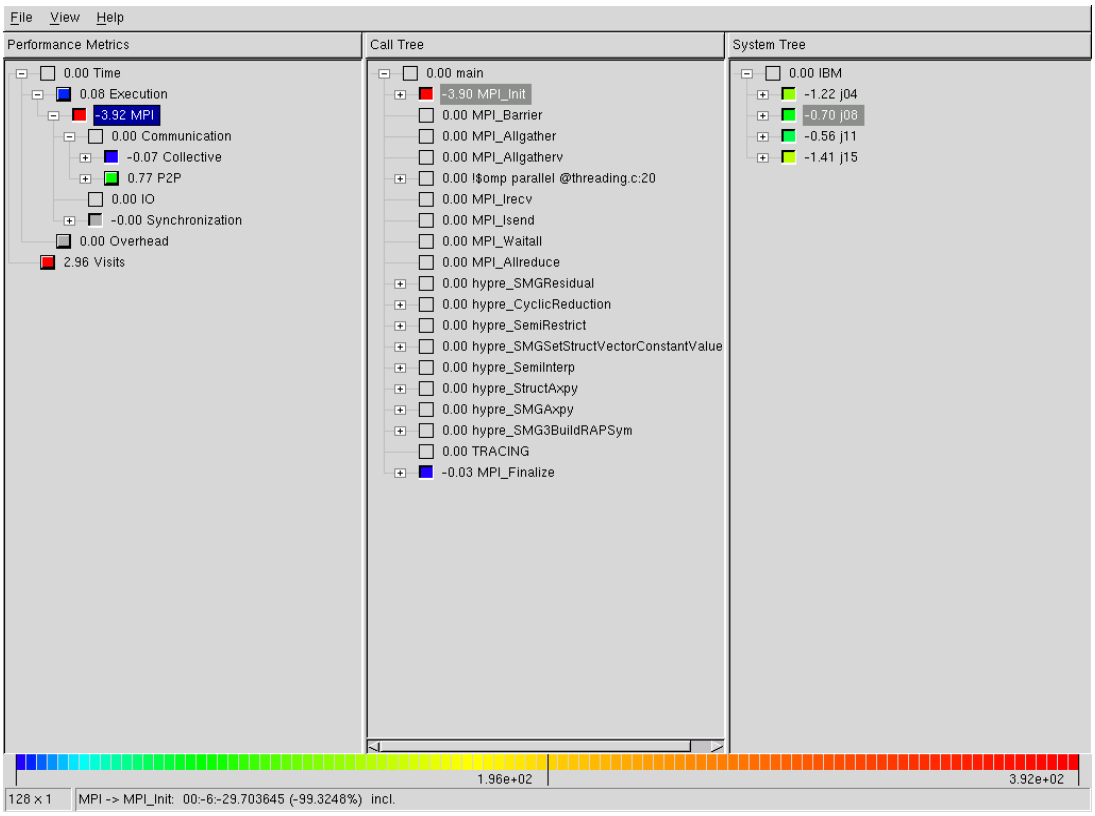


Figure 4: SMG2000 Diff 32×1 vs. 128×1

Increasing the number of MPI processes increases the time for running MPI, especially the time for MPI_Init was strongly increased (Figure 4, left). The reduction in P2P time is a little bit strange, because normally more MPI processes need more time for P2P communication. The improvement is generated by several Waitall regions, where the bigger run needs less time. So it is recommended to examine the 32×1 run at these regions.

The introduction of OpenMP multithreading (Figure 5) improves the time for running MPI and it increases the time of OpenMP and Idle threads in expected ways. All in all the improvement for running MPI improves the execution time of the whole program.

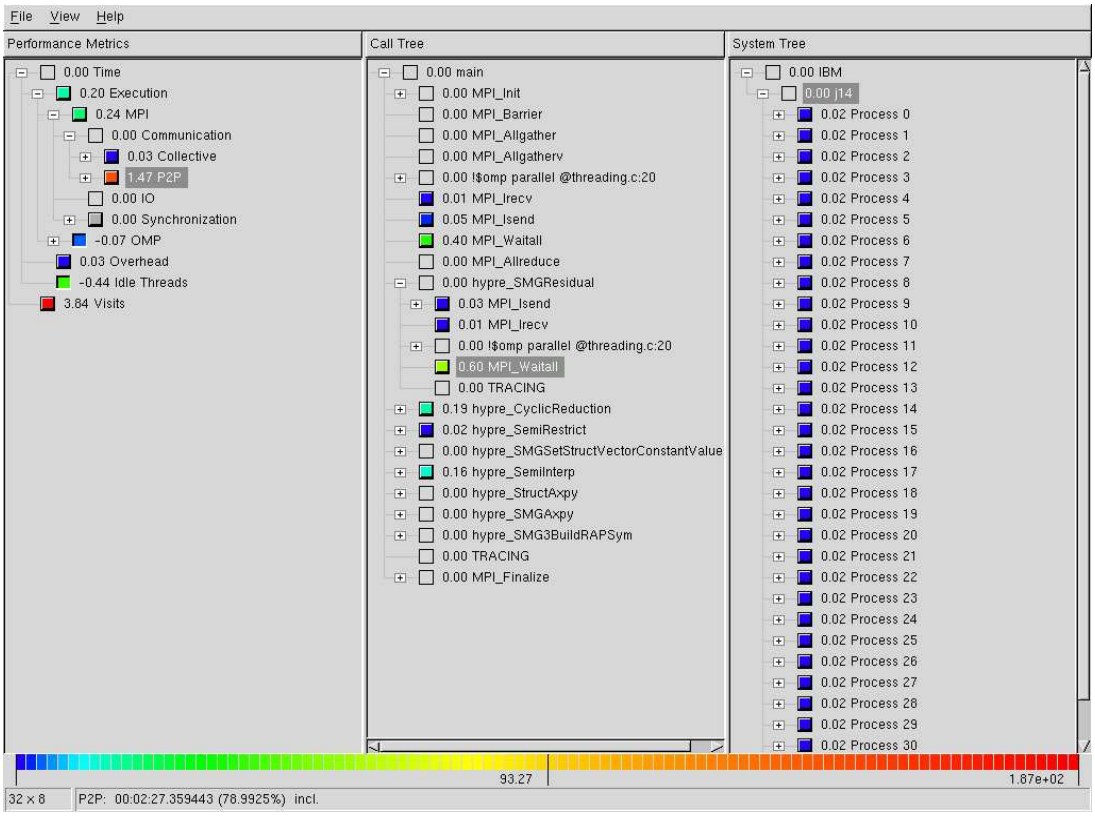


Figure 5: SMG2000 Diff 32×1 vs. 4×8

CP2K

General

Due to the very large size of CP2K and its complicated building procedures, KOJAK fails to instrument this application using the normal procedures. But there is another way to get an useful trace. The application possesses a time wrapper routine for all significant routines. So it is possible to instrument this timer function manually with a call to the `elg_user_start` and `elg_user_end` routines. After that, the user only has to link with the MPI and PAPI libraries and run the executable. However, this strategy only works for MPI applications.

Building the Code

1. Download the application.

2. Edit the file AIX-PowerPC_POWER4.popt in directory "arch". The following configuration was used.

```
PERL     = perl
CC       = cc
CPP      = /usr/ccs/lib/cpp
FC       = mpxlf90_r -qsuffix=f=f90
LD       = mpxlf90_r
AR       = ar -r -X64
DFLAGS   = -D__AIX -D__ESSL -D__FFTSG -D__FFTESSL\
```

```
            -D__parallel -D__BLACS -D__SCALAPACK -D__KOJAK
CPPFLAGS = -C  $(DFLAGS) -P
FCFLAGS  = -O2 -q64 -qarch=pwr4 -qcache=auto -qtune=pwr4
LDFLAGS  = $(FCFLAGS) -bnoquiet\
            -bmaxdata:0x800000000  -bmaxstack:0x80000000

LIBS     = -L /usr/local/beta/kojak-2.2b2/lib/64 -L. -lfmpi \
            -lelg.mpi -lmpi \
            -L /usr/local/beta/papi-3.0.8/lib -lpapi64 -lpmapi\
            -lc -lscalapack -lblacssmp -lpesslsmp \
            -llapack -lessl -lmass -lhm
```

3. Instrument the file timings.F in directory "src" with:

```
timings.F:382:
#if defined(__KOJAK)
      mytag=TRIM(timer_env%routine_name)
      mytag(20:20)=CHAR(0)
      call ELG_User_start(mytag,"",-1)
#endif

timings.F:485:
#if defined(__KOJAK)
      mytag=TRIM(timer_env%routine_name)
      mytag(20:20)=CHAR(0)
      call ELG_User_end(mytag,"",-1)
#endif
```

4. Change into directory "makefiles" and install with `make popt`. This will produce an executable file.

5. Change into directory "tests/QS" and execute the application with e.g.,
   `llrun -p 4 ../../exe/AIX-PowerPC_POWER4/cp2k.popt H2O-32.inp`
   on JUMP.

Results

Figure 6 shows the result of tracing and analyzing with KOJAK. 70 percent of the execution time is used for executing different functions in the application (Figure 6, left) . The routine dominating the execution is cp_fm_syevx (Figure 6, middle). The work done by this routine is well distributed on the different processes, because every process needs 1.3 percent of the run-time for executing this routine (Figure 6, right). The other 30 percent are distributed across the other routines.
26 percent of the execution time is used for MPI communication and further 3 percent are required for MPI_Init and MPI_Finalize. A total of 30 percent of the execution time is needed for using MPI.
The communication time is divided into 11.5 percent for collective operations and 14.4 percent for P2P operations. The 14.4 percent of the P2P operations are generated by 4 Waitall operations in the program. The collective time is divided into 4.8 percent for Wait at N×N, 1.4 percent for Late Broadcast and 5.3 percent for other event patterns.
So the major problems of this application are the Waitall operations and the Wait at N×N. Otherwise the CP2K application is well balanced.

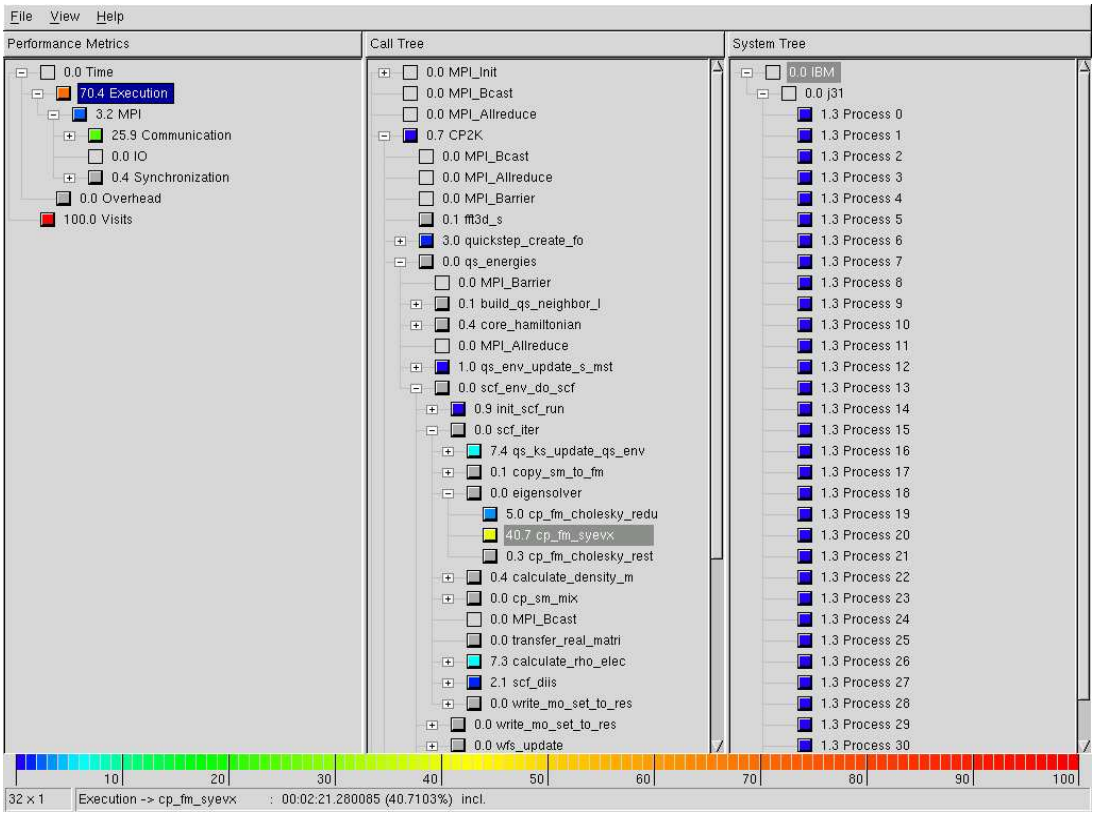


Figure 6: CP2K visual result with KOJAK

Fortran MPI testsuite

In addition to my work on evaluating KOJAK's usability for large applications, I worked on enhancing KOJAK's MPI testsuite (which is implemented in C) with additional Fortran testcases. This section describes the status of this work.

The Fortran testsuite supports testing of MPI point to point, nonblocking communication completion, and collective operations. Also, a wide variety of datatypes are supported. At the moment it runs only on an Intel Linux machine with the Intel Fortran compiler 9.0. Other systems were not tested.

Only systems and compilers which provide Fortran2000 commandline access are supported. As a workaround a preprocessor directive can be inserted in the source code, which switches to an internal read of the input parameters from a temporary file instead of the command line. This file could be created by piping the configurations from the testsuite run script. This option is not yet available.

Every test case contains a main program and a module with all necessary routines made available by generic interfaces.

There also some differences between the Fortran and the C testsuite. For example, Fortran supports no buffer type casting, so that a `select case` statement is used to switch between the different supported types. Also Fortran possesses datatypes different from C. As a consequence, the C MPI datatype `MPI_WCHAR` is not supported.
The MPI datatype `MPI_TYPE_VECTOR` is created in a different way. Based on an `INTEGER` vector with the size of the number of used elements, the vector is created with the half of the elements and with a stride of 2.
The implementation of some completion operations was changed because of a segmentation fault while

testing. Additional to the C version, routines for the reduction operations of non-standard datatypes are implemented and used. But at the moment, there are some complications with the combination of datatype `MPI_TYPE_VECTOR` and reduction operations like `MPI_ALLREDUCE`. The problem is that the reduction routine needs the whole `INTEGER` vector, but the distributing routine only needs some elements of the type vector. Either the distribution or the reduction operation works correctly. Currently, there is no solution so that both parts work correctly.

specifier	options
datatype	int, float, double, complex, char, logical, struct, vector
elements	something greater 0
tag	integer value (optional)
root	integer value for the root process
sendoperation	send, bsend, rsend, ssend, isend, ibsend, irsend, issend, sendrecv
recvoperation	recv, irecv
completion operation	test, testany, testsome, testall, wait, waitany, waitsome, waitall
collective operation	barrier, bcast, gather, gatherv, scatter, scatterv, allgather, allgatherv, alltoall, alltoallv, alltoallw, reduce, allreduce, reduce_scatter, scan, exscan

Table 2: Testsuite configuration

For running the tests use the following commands:

- ptp -e `elements` -g `tag` -s `sendoperation` -r `recvoperation` -t `datatype`
- ptpi -e `elements` -g `tag` -w `completion operation` -t `datatype`
- coll -e `elements` -o `collective operation` -r `root` -t `datatype`

with the various options shown in table 2.

It is recommended to specify the datatype in the last position, otherwise the execution with the datatype vector will fail.
Possibly, the execution may fail if the root process is unequal to 0. This case must be investigated later.

Conclusion

In this work, I successfully used KOJAK to investigate the performance of three large, real-world applications. The instrumentation of the codes did not work as described in the documentation of KOJAK. However, simple workarounds could be used to get around these problems. This information should be added to the documentation.

Future Work

The SMG2000 benchmark uses topology constructs, so KOJAK can be used to investigate this benchmark with its new topology feature.
The testsuite has to be tested further and ported to more platforms.

Acknowledgements

This work was carried out during the Summer Student's Programme for Scientific Computing at Research Centre Juelich. I would like to thank the organizer Dr. R. Esser for enabling me to participate in the program and all staff of ZAM for the excellent support. I would also like to thank my supervisor Dr. B. Mohr for his excellent support even while traveling.

References

1. EPILOG
   http://www.fz-juelich.de/zam/kojak/epilog/.
2. KOJAK
   http://www.fz-juelich.de/zam/kojak/ .
3. OPARI
   http://www.fz-juelich.de/zam/kojak/opari/ .
4. PAPI
   http://icl.cs.utk.edu/papi/.
5. The ASCI Purple Benchmarks - The UMT Benchmark Code
   http://www.llnl.gov/asci/purple/benchmarks/limited/umt/ .
6. The ASCI Purple Benchmarks - The SMG2000 Benchmark Code
   http://www.llnl.gov/asci/purple/benchmarks/limited/smg/ .
7. VAMPIR
   http://www.pallas.de/e/products/.
8. CP2K
   http://cp2k.berlios.de/.

Online-Visualisierung und Steuerung geodynamischer Simulationsrechnungen

Dirk Elbeshausen

Institut für Geophysik der Westfälischen Wilhelms-Universität Münster

e-mail: d.elbesh@uni-muenster.de

Zusammenfassung: Durch die stetige Weiterentwicklung von Algorithmen werden wissenschaftliche Simulationsrechnungen immer aufwendiger. Viele naturwissenschaftliche Disziplinen, darunter auch die Geodynamik, sind heutzutage auf massiv parallele Rechnerarchitekturen angewiesen. Da Rechenzeit auf solchen Hochleistungsrechnern nur begrenzt zur Verfügung steht, ist eine regelmäßige Kontrolle des laufenden Simulationsprogramms unerlässlich. Um diese auf eine effiziente Art zu ermöglichen, wurde eine Schnittstelle zur Kommunikation zwischen einer geodynamischen Simulationsrechnung und der entsprechenden Visualisierungssoftware entwickelt. Dabei wird die Visualisierungssoftware mit dem Simulationsprogramm gekoppelt und eine direkte Kommunikation ermöglicht. Dadurch können sämtliche Daten von der Simulation zur Visualisierung gesendet und dort graphisch dargestellt werden (*Online-Visualisierung*). Die Kommunikation zwischen beiden Seiten ermöglicht weiterhin eine deutliche Reduzierung des Datentransfers. Indem die Visualisierung gezielt von der Simulation Daten anfordert, kann dort vor dem Versenden der Daten eine Selektion durchgeführt werden. Somit werden nur die Daten transferiert, die zur Visualisierung momentan erforderlich sind. Des weiteren wurde dem Benutzer die Möglichkeit eröffnet, direkten Einfluss auf die Simulationsrechnung zu nehmen und ausgewählte Parameter noch während des Rechenvorgangs zu verändern (*Computational Steering*).

Einleitung

Wissenschaftliche Simulationsrechnungen werden dank stetig verbesserter Algorithmen immer aufwendiger. Zu einer der größten Herausforderungen der Computational Physics zählt heute die Untersuchung geodynamischer Fragestellungen. Erst Höchstleistungsrechner wie der IBM p690-Cluster JUMP [8] ermöglichen überhaupt intensive Studien zur Entstehung des Erdmagnetfeldes, der thermischen Entwicklung von Planeten oder der Entstehung und Entwicklung von Plattentektonik. Die dynamischen Prozesse im Erdinneren sind sehr komplex, so dass extrem anspruchsvolle und hochauflösende Simulationsrechnungen notwendig sind. Am Institut für Geophysik der Universität Münster [1] wird u.a. erforscht, wie Konvektionsströmungen im äußeren Erdkern das Erdmagnetfeld generieren, wie die Konvektion im Erdmantel zur Entstehung der Plattentektonik führt oder welche Einflüsse dafür verantwortlich sind, dass die Erde sich von den anderen Planeten unseres Sonnensystems so stark unterscheidet. Da sich das Innere unseres Planeten geophysikalischen Messungen weitestgehend entzieht, sind numerische Simulationsrechnungen unerlässlich. Solche Programme werden in Münster entwickelt und verwendet. Die entsprechenden Rechnungen sind allerdings so aufwendig, dass sie nur unter dem Einsatz massiv paralleler Rechner bewerkstelligt werden können. Die Analyse der Ergebnisse erfolgt zumeist in Verbindung mit der Visualisierung, da nur so auch die wichtigen kleinskaligen Strukturen erfasst und untersucht werden können. Die entsprechende Visualisierungssoftware, ebenfalls in Münster entwickelt, wird jedoch auch

eingesetzt, um Zwischenergebnisse zu visualisieren und somit die Simulationsrechnung zu kontrollieren. Dies erfolgte bislang *offline*, d.h., dass keine Verbindung zwischen Visualisierung und Simulation vorliegt. Das Simulationsprogramm erzeugt in regelmäßigen Abständen Dateien, in die entsprechende Zwischenergebnisse binär geschrieben werden. Zur Visualisierung müssen diese Dateien dann auf den lokalen Rechner transferiert und anschließend umformatiert werden. Da diese Vorgehensweise arbeits- und zeitintensiv ist, wurden im Rahmen dieser Arbeit die o.g. Programme um die Möglichkeit erweitert, die Datenübertragung zukünftig *online* erfolgen zu lassen. Dabei wird die Visualisierungssoftware mit dem Simulationsprogramm gekoppelt und eine direkte Kommunikation ermöglicht. Dazu wurde eine Schnittstelle entwickelt und in die Visualisierungssoftware und den Simulationscode integriert. Somit können sämtliche Daten von der Simulation zur Visualisierung gesendet und dort graphisch dargestellt werden (*Online-Visualisierung*). Die Kommunikation zwischen beiden Seiten ermöglicht weiterhin eine deutliche Reduzierung des Datentransfers. Da Informationen zu den benötigten Daten zur Simulation versendet werden, kann dort eine Selektion stattfinden, noch bevor die Daten versendet werden. Dadurch werden nur die zur Visualisierung momentan erforderlichen Daten transferiert. Des weiteren wurde dem Benutzer die Möglichkeit eröffnet, direkten Einfluss auf die Simulationsrechnung zu nehmen und ausgewählte Parameter noch während des Rechenvorgangs zu verändern (*Computational Steering*).

Bei der Entwicklung und Integration dieser Schnittstelle waren folgende Faktoren ausschlaggebend:

- Benutzerfreundlichkeit:
  Intensive Kenntnisse von Netzwerken und Server-Architekturen sollen nicht nötig sein, um diese Schnittstelle verwenden zu können.
- Portabilität:
  diese Schnittstelle muss auf den unterschiedlichsten Rechnerarchitekturen laufen, sowie auch mehrere Verbindungsmöglichkeiten unterstützen, um die Daten von jedem Rechner transferieren zu können.
- Modularisierung:
  Die Schnittstelle sollte möglichst allgemein gehalten werden, um sie zukünftig auch sehr einfach in andere Simulationscodes integrieren zu können.
- Effektivität:
  ein wesentlicher Aspekt war vor allem, die Interaktion zwischen Visualisierung (*Server*) und Simulation (*client*) so ablaufen zu lassen, dass die Simulation durch die Kommunikation möglichst nicht gestört wird.

Grundlagen

Diese Arbeit beschäftigt sich im wesentlichen mit der Kopplung, d.h. dem Zusammenwirken unterschiedlicher Programme und Bibliotheken auf verschiedenen Ebenen. Daher sollen an dieser Stelle die entsprechenden Komponenten kurz vorgestellt werden.

Neben der in Münster entwickelten Visualisierungssoftware *Sphere* [3] und den ebenfalls dort entstandenen Simulationscodes [7] ist hier vor allem die am ZAM [10] entwickelte Kopplungsbibliothek *VISIT* [5][6] zu nennen.

Das Simulationsprogramm

Am Institut für Geophysik der Universität Münster [1] beschäftigt man sich seit langem mit geodynamischen Simulationsrechnungen in sphärischen Geometrien. Das verwendete, sogenannte *"Cubed-Sphere-Gitter"* [7] lässt realistischere Untersuchungen zu, als dies mit Simulationen in einer kartesischen Box

der Fall ist. Allerdings bringt das deutlich komplexere sphärische Gitter auch eine erheblich aufwendigere Visualisierung sowie zeitintensive Berechnungen mit sich. Die Rechnungen verlaufen iterativ, was die Kopplung mit der Visualisierung erleichtert. Als Verbindungszeitpunkt, das ist der Zeitpunkt, an dem der Nachrichten- oder Datenaustausch stattfindet, wurde das Ende eines berechneten Zeitschrittes gewählt. In regelmäßigen Abständen, typischerweise alle 100-200 Zeitschritte, wird von jedem Prozessor ein sogenanntes *Dump-File* erzeugt. Dabei handelt es sich um eine Datei, in die Ergebnisse des aktuellen Zeitschritts binär geschrieben werden. Ursprünglich dazu konzipiert, um abgebrochene Rechnungen fortsetzen zu können, wurden diese Dateien bislang auch zur Visualisierung der Zwischenergebnisse verwendet. Da die Visualisierungssoftware und das Simulationsprogramm in der Regel auf unterschiedlichen Rechnern gestartet werden, ist ein Transfer der *Dump-Files*, in der Regel über eine SSH-Verbindung, notwendig. Je nach Auflösung des verwendeten Gitters handelt es sich hierbei um ein Datenvolumen von bis zu 1 GB, so dass die Übertragung selbst bei schnellen Netzverbindungen einige Zeit in Anspruch nimmt. Weitere Nachteile der Verwendung von *Dump-Files* zur Visualisierung sind:

- Beim Datentransfer werden auch nicht benötigte Daten kopiert. Zur Visualisierung werden in der Regel nur 4 Felder (ein skalares Feld sowie die drei Komponenten des Vektorfeldes) benötigt. In den Datensätzen befinden sich jedoch viele weitere Felder, die mit transferiert werden.
- Die Daten in den *Dump-Files* liegen in 8-Byte (double precision) Genauigkeit vor. Da zu Visualisierungszwecken einfache Genauigkeit (4-Byte) aber völlig ausreicht, wird erneut ein größeres Datenvolumen transferiert als eigentlich nötig.
- Die für das Transferieren der Daten notwendigen Schritte (Aufbau einer ssh- oder TCP-Verbindung, möglicherweise noch der Aufruf eines Proxy-Servers sowie das eigentliche Kopieren) müssen z.B. zum Visualisieren von Daten anderer Zeitschritte jedesmal wiederholt werden. Zudem müssen Analysetools oder die Visualisierungssoftware in der Regel immer wieder mit den aktuellen Daten neu gestartet werden.
- Die oben beschriebenen Datensätze werden nur in bestimmten Intervallen (s.o.) gespeichert. Die Daten, die über das Netz transferiert werden, sind zumeist also nicht aktuell.

Diese Punkte zeigen die Vorteile auf, die durch eine *Online-Visualisierung*, d.h. einen direkten Datenaustausch zwischen Simulation und Visualisierung, entstehen.

Sphere

Um die sehr großen Datensätze, die aus diesen geodynamischen Simulationsrechnungen resultieren, schnell und flexibel zu Visualisierungszwecken nutzen zu können, wurde in Münster eine Software entwickelt (*Sphere* [3], siehe Abb. 1). Da diese Software unmittelbar für die im Simulationsprogramm verwendete Cubed-Sphere-Geometrie [7] geschrieben worden ist, ist das Einlesen und Handling der Daten effizienter und flexibler als bei Verwendung herkömmlicher, kommerzieller Softwarepakete. Auch die Handhabung des Programms ist benutzerfreundlicher, da die Software zur Visualisierung und Analyse dieser speziellen Datensätze entwickelt wurde.

Zur Visualisierung der Daten kann auf zahlreiche Verfahren zurückgegriffen werden, wie z.B. Oberflächen- und Isoflächendarstellungen, Ausschnittsberechnungen, Schnittflächen, Volumen-Rendering, Vektorpfeile, Stromlinien, Streamer oder Tracer. Jedes Verfahren kann der Benutzer entsprechend seiner Präferenzen anpassen, was zu einer höheren Flexibilität der Software führt. Zusätzliche Veränderungen am Programm können über eine Plugin-Schnittstelle vorgenommen werden. Die graphische Oberfläche sorgt für eine einfache Bedienung. Weitere Merkmale, wie z.B. eine integrierte Encodier-Funktion zum Erzeugen von Videos oder ein Animationsmenü sorgen für eine spürbare Arbeitserleichterung für die Anwender.

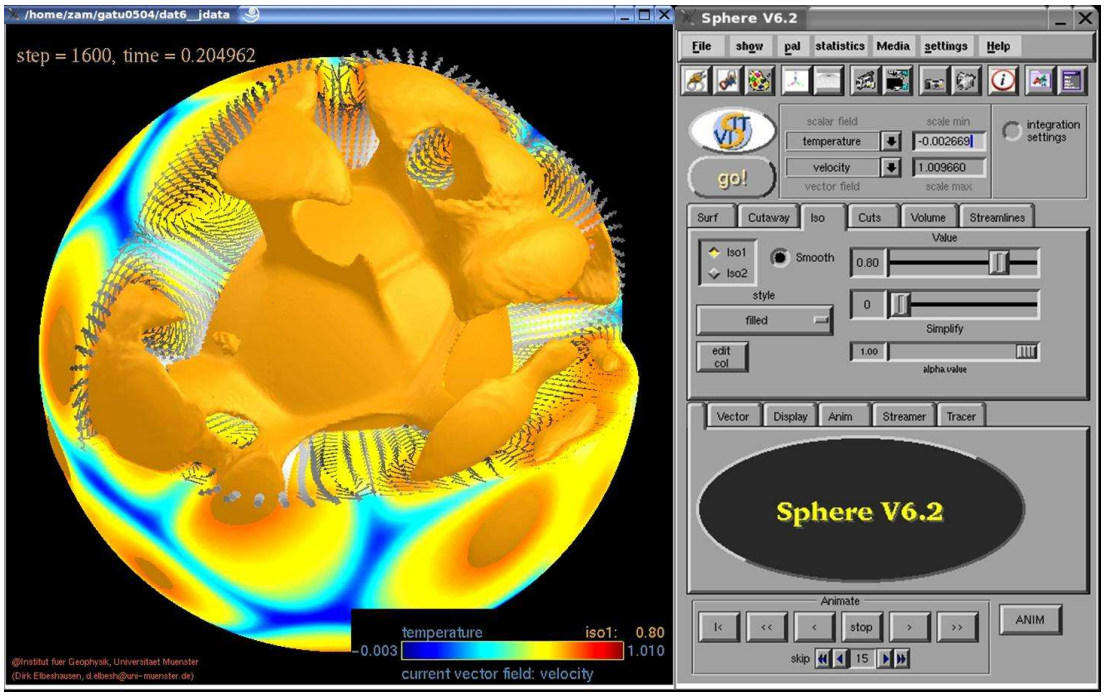


Abbildung 1: Die Visualisierungssoftware *"Sphere"*
Links: Die Visualisierung (hier zu sehen: Isoflächen kombiniert mit Ausschnittsberechnungen und Vektorpfeilen). In diese Darstellung kann der Benutzer unter Verwendung der Maus interaktiv hineinzoomen. Auch Rotation und Translation sind möglich. Über die graphische Benutzeroberfläche (rechts) können zahlreiche Visualisierungsverfahren miteinander kombiniert oder angepasst werden. Auch auf zusätzliche Hilfsfunktionen (wie z.B. Erstellen von Videos) kann hier zugegriffen werden.

Visit

Um dem Anwender den Aufbau einer Kommunikation zwischen Rechnern zu erleichtern, wurde am ZAM [10] das ***VIS**ualization **I**nterface **T**oolkit (VISIT)* [5][6] entwickelt. Diese Kopplungsbibliothek beinhaltet vor allem Funktionen zum Auf- oder Abbau einer Verbindung zwischen Simulation und Visualisierung und zum Datenaustausch. Durch eine Integration in *Unicore* [9] (in Arbeit) wird die Authentifizierung sehr einfach. *VISIT* basiert auf einem einfachen Client-Server-Modell. Dies bedeutet, dass kein zusätzlicher Zentral- oder Datenserver in die Kommunikation eingebunden werden muss.

Um den Verbindungsaufbau für den Anwender so einfach wie möglich zu gestalten, kann ein *Seap-Server*[1] als dritte Instanz verwendet werden. Dort werden Hostname und Portnummer zusammen mit einem Service-Namen und Passwort hinterlegt. Simulationsprogramm und Visualisierungssoftware können nun von dort die Informationen abrufen, sofern ihr Service-Name und Passwort damit übereinstimmen. Die Verwendung des *Seap-Servers* stellt eine einfache und effiziente Möglichkeit dar, auf der Simulationsseite festzustellen, ob die entsprechende Visualisierungssoftware bereit zum Datenempfang ist (siehe Abb. 2). Sobald der *Seap-Server* die entsprechenden Verbindungsdaten an die Simulation gesendet hat, kann der eigentliche Verbindungsaufbau stattfinden.

[1] seap = service announcement protocol

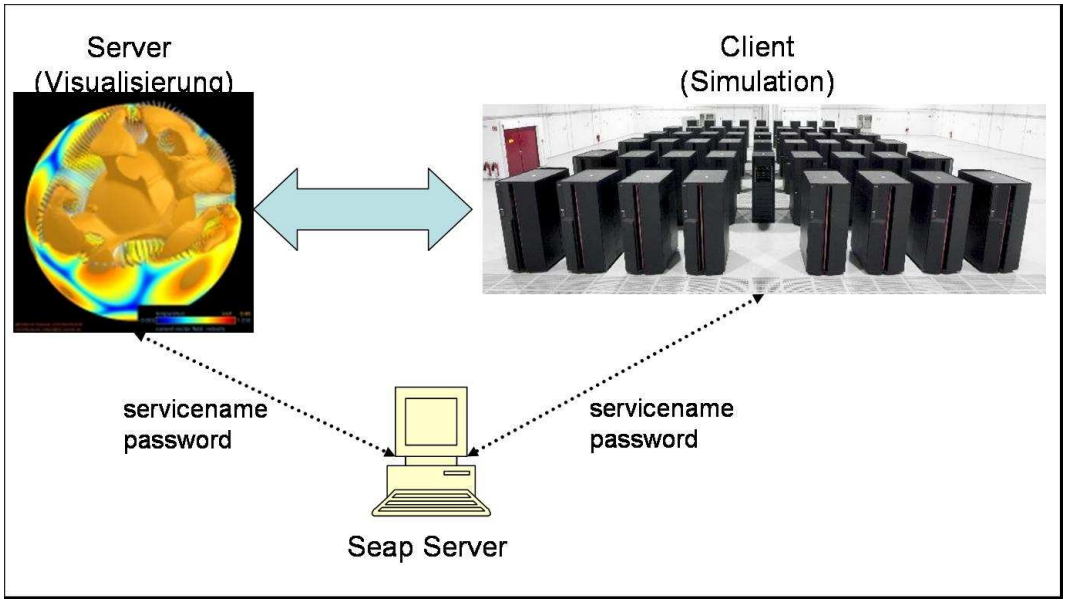


Abbildung 2: Verbindungsaufbau mittels SEAP-Server
Der Seap-Server ist eine dritte Instanz und dient einem flexibleren Verbindungsaufbau. Hostname und Portnummer werden hier von der Visualisierung hinterlegt und können mit einem Service-Namen und Passwort von der Simulation abgerufen werden.

Als Übertragungsprotokoll werden TCP/IP-Sockets verwendet, da sie auf allen Plattformen vorhanden sind und damit Portabilität gewährleisten. *VISIT* ermöglicht eine bidirektionale, transparente Übertragung von Daten zwischen *Client* (Simulation) und *Server* (Visualisierung). Durch entsprechende Funktionen kann ein Datenaustausch auf unterschiedliche Verbindungsarten erfolgen. Die schnellste Verbindungsart ist eine direkte TCP/IP-Verbindung. Hierbei werden die Daten unmittelbar von Socket zu Socket übertragen. Da Visualisierungssoftware und Simulation in der Regel auf unterschiedlichen Rechnern gestartet werden, verhindert oft eine Firewall diese Art der Kommunikation. Deshalb bietet *VISIT* zusätzlich auch die Möglichkeit, die Verbindung über einen *ssh-Tunnel* aufzubauen. Hierbei werden die Daten verschlüsselt und unter Verwendung eines Proxyservers übertragen (siehe Abb. 3).

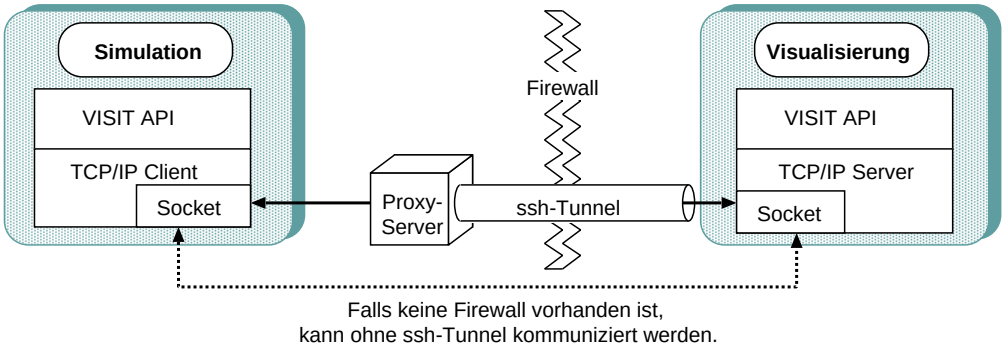


Abbildung 3: Schematische Darstellung der von *VISIT* unterstützten Verbindungsarten:
Ein schneller Datentransfer direkt über TCP/IP-Sockets ist allerdings nur möglich, wenn keine Firewall die Übertragung verhindert. Sollte das der Fall sein, kann der Transfer verschlüsselt über einen ssh-Tunnel erfolgen. (Abb. entnommen aus [2])

Verbindungskonzepte - Kopplung der Programme

Ein wesentlicher Bestandteil dieser Arbeit war, durch Verwendung von *VISIT* eine Kopplung zwischen *Sphere* und dem Simulationsprogramm zu herzustellen. Dazu war es notwendig, in beiden Programmcodes - dem der Visualisierungssoftware und auch dem des verwendeten Simulationsprogramms - Modifikationen bzw. Erweiterungen vorzunehmen. Dabei wurde darauf geachtet, die Eingriffe in dem Simulationscode so übersichtlich und minimal wie möglich zu gestalten. Die Integration dieser Schnittstelle in weitere Simulationsprogramme kann daher zukünftig sehr einfach und ohne großen Arbeitsaufwand durchgeführt werden.

In den folgenden Abschnitten wird erläutert, wie diese Schnittstelle in die einzelnen Codes integriert wurde.

Integration in den Simulationscode

Die Simulation sollte durch die Schnittstelle in ihrem Ablauf möglichst wenig beeinträchtigt werden. Sie kann als *Client* aufgefasst werden, da sie Anfragen an die Visualisierung dann sendet, sobald sie dazu bereit ist. Die Visualisierung übernimmt die Rolle eines Servers, indem sie auf Anfragen reagiert.

Solange noch keine Verbindung zur Visualisierung besteht, wird auf der Client-Seite nach jedem Zeitschritt Kontakt zum Seap-Server aufgenommen. Sobald über diesen eine Verbindung zur Visualisierung hergestellt wurde, beginnt der eigentliche Informationsaustausch. Über eine direkte Anfrage an den Server wird zunächst überprüft, ob die Visualisierung Daten senden möchte oder anfordert. Anschließend werden vom Client die erforderlichen Daten zusammengestellt, in einem Puffer abgelegt und zusammen an die Visualisierung gesendet. Dadurch wird gewährleistet, dass der Client nur minimale Zeit zum Versenden der Daten aufwenden muss (siehe “Ergebnisse” auf S. 46).

In dem Puffer werden nach jedem Zeitschritt zunächst wichtige Informationen zur Konfiguration der Simulation wie z.B. Randwerte oder Startparameter sowie zum Verlauf der Rechnung, u.a. die Nummer des aktuellen Zeitschritts oder die diffusive Zeit, geschrieben. Zusätzlich werden auch Statistiken, z.B. die mittlere Temperatur oder RMS-Geschwindigkeit, die für den aktuellen Zeitschritt berechnet wurden, im Puffer gesammelt. Diese Daten können dann später von der Visualisierung ausgewertet und automatisch mit Hilfe von 2D-Plots dargestellt werden. Ob weitere Daten in den Puffer geschrieben werden, hängt von der Anfrage der Visualisierung, und damit von der Entscheidung des Benutzers, ab. Werden Datenfelder, z.B. Temperatur, Viskosität oder Magnetfeldkomponenten, angefordert, so werden auf der Simulationsseite lediglich die benötigten Datenfelder von allen Prozessoren gesammelt (“gathering”) und ebenfalls in den Puffer geschrieben. Informationen zum Gitter (Koordinaten-, Normalen- und Abstandsvektoren) werden nur einmal, unmittelbar nach Verbindungsaufbau, versendet, da die Gitterstruktur zeitlich konstant ist. Der Ablauf ist in Abb. 4 noch einmal verdeutlicht.

Kann der Datenaustausch mit dem Server nicht mehr erfolgen (z.B. weil die Visualisierungssoftware beendet wurde), wird auf der Simulationsseite die Verbindung zur Visualisierung abgebaut. Danach erfolgt wiederum in jedem Zeitschritt die Kontaktaufnahme mit dem Seap-Server. Dadurch ist gewährleistet, dass der Informations- oder Datenaustausch schnell fortgesetzt werden kann, sobald die Visualisierung wieder ihre Daten auf dem Seap-Server hinterlegt hat.

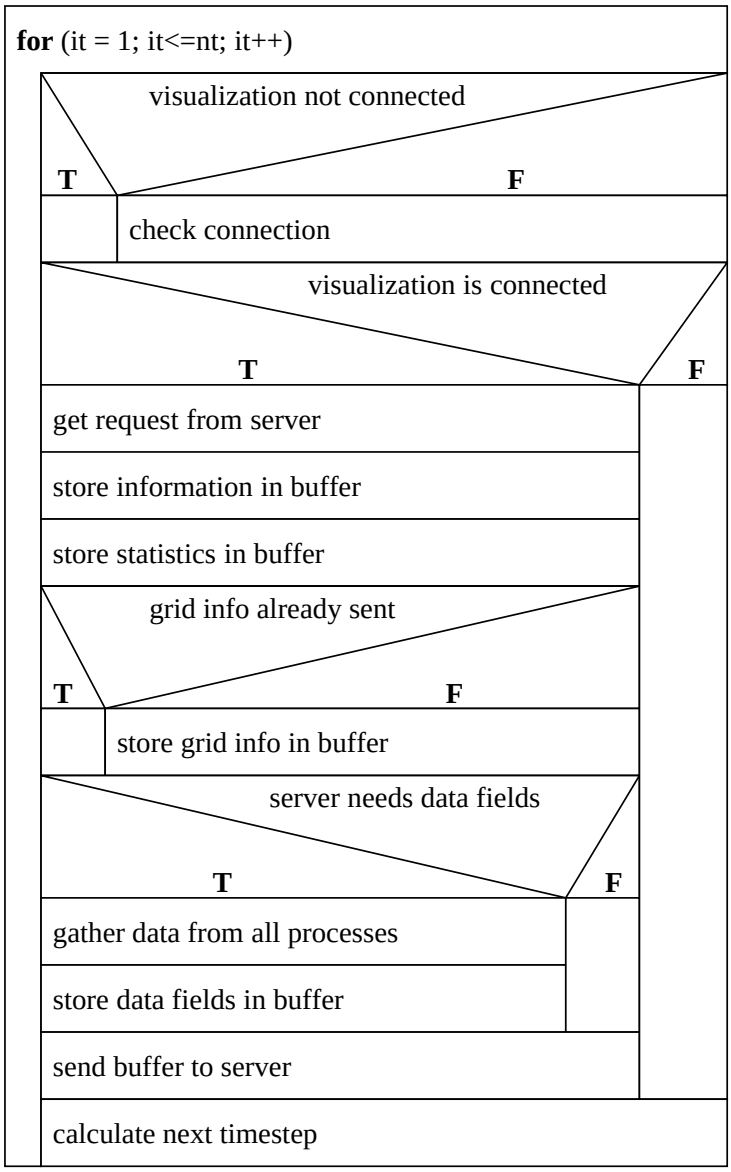


Abbildung 4: Nassi-Shneiderman-Struktogramm der Abläufe innerhalb der Simulationsrechnung

Integration in Sphere

Die Implementierung der Schnittstelle in die Visualisierung gestaltet sich wesentlich aufwendiger. Die Schwierigkeit besteht vor allem darin, den genauen Zeitpunkt zu ermitteln, an dem die Simulation bereit zum Empfangen oder Senden von Daten ist. Die einfachste Möglichkeit wäre, in regelmäßigen Abständen (z.B. alle 100ms) Anfragen an die Simulation zu senden, um zu überprüfen, ob diese reagiert. Dieser Weg ist allerdings nicht sinnvoll, da er auf der Visualisierungsseite zu Performance-Verlusten führt. Dieses Problem konnte umgangen werden, indem die Programmstruktur von *Sphere* ausgenutzt wurde. Die meisten Prozesse von *Sphere* werden in einer sogenannten *Eventloop* ausgeführt. Das Programm befindet sich dabei in einer Endlosschleife, bis es vom Benutzer beendet wird. Die notwendigen Aktionen innerhalb dieser Schleife werden mittels *Callback-Funktionen* ausgeführt. Dies sind Funktionen, die nur dann aufgerufen werden, wenn das dazugehörige Ereignis ausgelöst wurde. Solch ein Ereignis kann zum Beispiel das Bewegen der Maus, Drücken einer Taste auf der Tastatur oder aber eine Statusänderung auf

einem TCP/IP-Socket sein.

Um auf die Anfragen der Simulation schnell reagieren zu können, wurde eine Socket-Callback-Funktion verwendet. Hierbei wird der Socket der Visualisierung im Hintergrund überwacht. Sobald dort Nachrichten ankommen, wird die oben genannte Callback-Funktion ausgeführt. Diese sorgt dann für den Datenaustausch. Somit wird die Visualisierung nur dann in ihrem Ablauf beeinträchtigt, wenn ein Datenaustausch tatsächlich stattfinden kann.

Ergebnisse der Performance-Analyse

Der Datenaustausch sowie die Kommunikation mit Seap-Server und Socket der Gegenseite sollten möglichst wenig Zeit beanspruchen. Dies gilt insbesondere für die Simulation. Daher wurde eine Laufzeit-Analyse durchgeführt, um festzustellen, wie groß der Zeitbedarf für die Kommunikation ist. Grundlage der Analyse war eine Simulation mittlerer Auflösung (200.000 Gitterpunkte). Die Ergebnisse sind in Abb. 5 und 6 aufgezeigt.

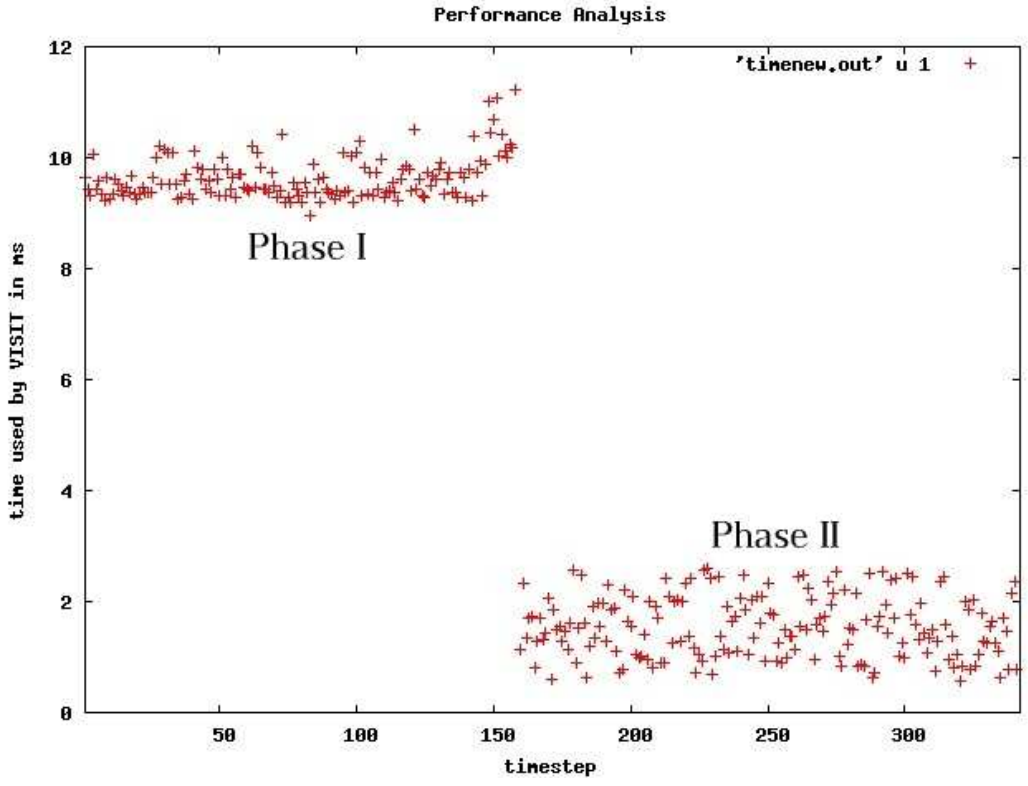


Abbildung 5: Zeitverbrauch für *VISIT* während der Simulationsrechnung
Phase I: Visualisierung ist nicht online
Phase II: Visualisierung ist online, es werden *keine* Datenfelder angefordert

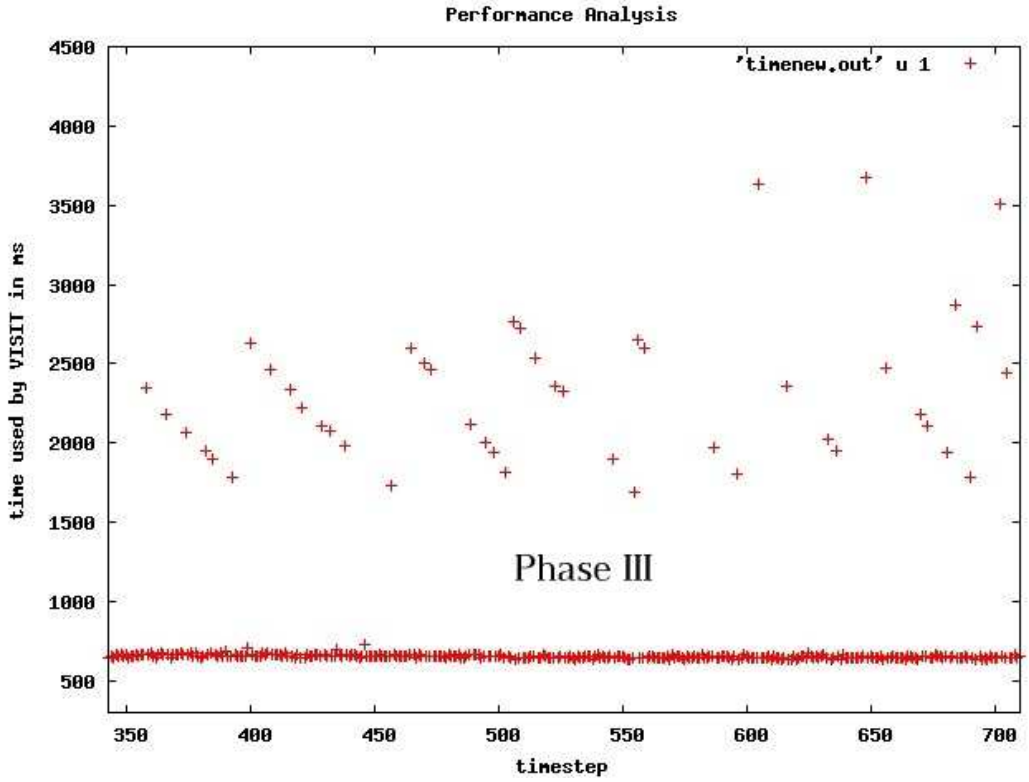


Abbildung 6: Zeitverbrauch für *VISIT* während der Simulationsrechnung
Phase III: Visualisierung ist online, es werden 4 Datenfelder angefordert

Der Test lässt sich in drei Phasen einteilen:

- Phase I:
  die Visualisierung ist nicht online. Diese Information erhält die Simulation über den Seap-Server. Die Anfrage dauert etwa 9.5 ms.

- Phase II:
  die Visualisierung ist online - es werden jedoch **keine** Datenfelder angefordert. Hier empfängt die Simulation zunächst ein kleines, ca. 20 Byte großes, Informationspaket. Da in diesem Fall keine Datenfelder angefordert werden, findet hier lediglich das Puffern und Versenden von Laufzeitinformationen sowie Statistiken statt. Dieser gesamte Prozess nimmt ca. 1.2 ms in Anspruch.

- Phase III:
  die Visualisierung ist online - es werden **vier** Datenfelder angefordert. Zusätzlich zu den Schritten in Phase II werden hier die Datenfelder von den einzelnen Prozessen an den Master-Prozess gesendet, dort in den Puffer geschrieben und versendet. Wie viel Zeit diese Phase in Anspruch nimmt, hängt insbesondere von der Auflösung des Rechengitters und der Geschwindigkeit der Internetverbindung ab. In diesem Fall belief sich der Zeitverbrauch auf ca. 650 ms. Die vereinzelten Ausreißer in Abb. 6 können u.a. durch eine zusätzliche Belastung der Netzverbindung durch andere Prozesse erklärt werden.

Da die Simulation im Regelfall nur zu einem sehr geringen Anteil ihrer Gesamtlaufzeit mit der Visualisierung verbunden ist, ist vor allem der Zeitverbrauch von Phase I sehr wichtig. Die 9.5 ms, die hier

pro Zeitschritt benötigt werden, fallen angesichts der deutlich höheren Rechenzeit für einen Zeitschritt (zwischen 5 s und 20 min - je nach Auflösung und Zahl der verwendeten Prozessoren) nicht ins Gewicht.

Auch das Versenden der Statistiken und Laufzeitinformationen (Phase II) benötigt nicht wesentlich mehr Zeit als Phase I. Da hier nur etwa 100 Bytes verschickt werden, ist auch diese Phase als weitgehend unabhängig von der Netzgeschwindigkeit zu sehen.

Im Gegensatz zu den vorangegangenen Phasen liegt in Phase III ein relativ hohes zu transferierendes Datenvolumen vor (je nach Auflösung bis zu 80 MB). Dadurch spielt die Geschwindigkeit des Netzes hier eine entscheidende Rolle. Aus diesem Grund wurde dem Benutzer die Entscheidung überlassen, ob die Daten jeden Zeitschrittes oder nur des aktuellen übertragen werden sollen. Ansätze, um dieses Problem zu beheben, konnten aus Zeitgründen in dieser Arbeit nicht behandelt werden (mehr dazu im Ausblick auf S. 48).

Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurden bestehende Programme um die Möglichkeit erweitert, eine Kommunikation zwischen Simulation und Visualisierung herzustellen. Dadurch konnte eine Online-Visualisierung realisiert werden, die dem Benutzer die Möglichkeit bietet, die Simulation in nahezu Echtzeit (*online*) zu überwachen. Bei Bedarf kann nun über die Visualisierungssoftware das Simulationsprogramm gesteuert werden, indem Parameter des Simulationscodes über das Netz verändert werden können (*Computational Steering*). Dadurch kann ein Abbruch oder Neustart einer Simulationsrechnung oft verhindert und somit wertvolle Rechenzeit eingespart werden. Als ein Ergebnis dieser Arbeit ist in Abb. 7 ein Teil der Benutzeroberfläche zur Online-Visualisierung dargestellt.

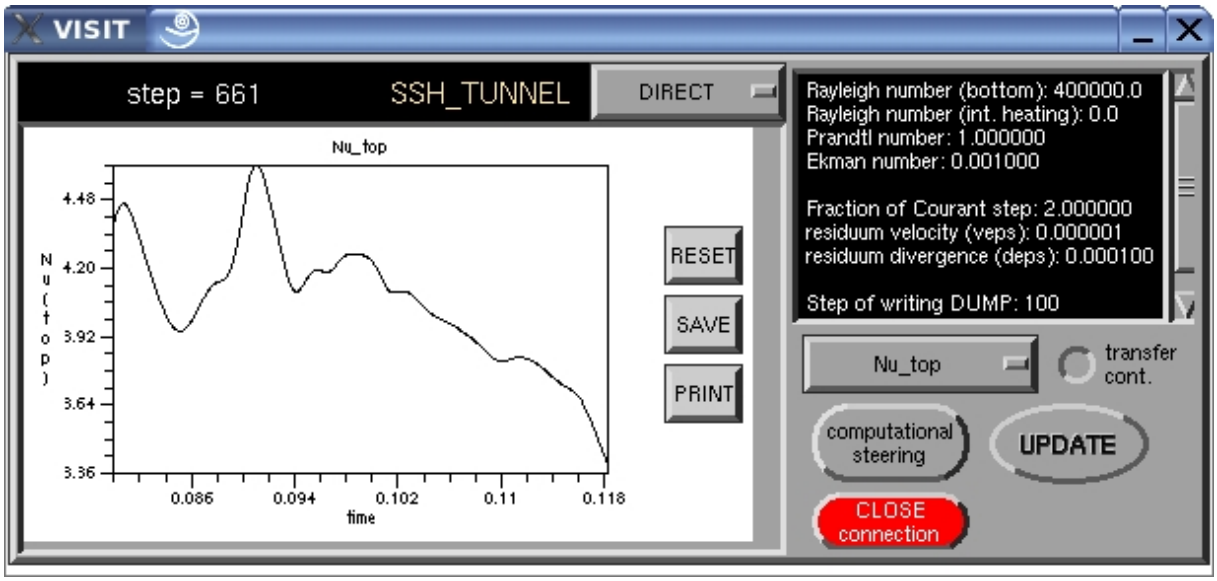


Abbildung 7: Die Benutzeroberfläche der Online-Visualisierung
Oben rechts werden Informationen zur Konfiguration der Rechnung angezeigt, oben links wird der aktuelle Zeitschritt eingeblendet. Im Hauptfenster kann der Benutzer sich wahlweise eine von bislang zehn Statistiken anzeigen lassen. Über die Schaltfläche *Update* entscheidet der Anwender selbst, wann die aktuellen Datenfelder angefordert werden sollen. Durch das Aktivieren des Kästchens *cont. transfer* werden die Daten automatisch nach jedem Zeitschritt übertragen.

Die Möglichkeit, Nachrichten vom Server zur Visualisierung zu versenden, konnte zusätzlich vorteilhaft verwendet werden. Indem der Client Informationen zu den vom Server benötigten Daten erhält und analysiert, kann er eine Datenreduktion vornehmen, so dass nur die benötigten Daten versendet werden. Dadurch konnte die für Kommunikation und Transfer benötigte Zeit auf ein Minimum reduziert werden. Werden lediglich Statistiken übertragen, so liegt der Zeitbedarf bei etwa 1.2 ms, so dass hier eine Kontrolle der Simulationsrechnung in Echtzeit möglich ist. Die zum Versenden von Datenfeldern benötigte Zeit hängt maßgeblich von der Größe der Datenfelder sowie der Geschwindigkeit der Internetleitung ab. Durch die verwendete Datenreduktion beschränkt sich das zu transferierende Datenvolumen in der Regel auf wenige MB. Dank der hohen Datenübertragungsraten innerhalb des Wissenschaftsnetzes, an das die meisten Forschungseinrichtungen angeschlossen sind, liegt der Zeitaufwand der Datenübertragung meist unterhalb einer Sekunde. Damit kann auch die Online-Visualisierung in der Regel nahezu in Echtzeit erfolgen.

Bei extrem hochauflösenden Rechnungen kann es allerdings Fälle geben, in denen das Datenvolumen größer ist. Um die Performance vor allem hinsichtlich solcher Simulationsrechnungen zu verbessern, könnten in Zukunft noch folgende Ansätze untersucht werden.

Eine einfache Möglichkeit wäre eine Datenreduktion auf der Simulationsseite. Dabei könnten z.B. nur die Werte jedes zweiten oder dritten Gitterpunktes an die Visualisierung gesendet werden [4]. Mit dem somit erzeugten gröberen Gitter geht allerdings auch ein Qualitätsverlust der Visualisierung einher. Dies ist in der Regel nicht hinnehmbar, da das Augenmerk in der Geodynamik oft auf den kleinskaligen Strukturen liegt.

Eine weitere Möglichkeit, die im Rahmen dieser Arbeit im Ansatz realisiert werden konnte, ist die Verwendung eines Datenserver-Modells. Hierbei werden die Daten nicht unmittelbar an die Visualisierung gesendet, sondern zunächst unformatiert auf der Festplatte gespeichert. Die zeitaufwendigen Prozesse wie das “gathern” und Umformatieren sowie Versenden der Daten übernimmt dann ein zusätzlicher Prozessor. Dadurch kann die Simulation entlastet werden. Jedoch dauert es effektiv länger, bis die Daten auf der Visualisierungsseite zur Verfügung stehen..

Danksagungen

Recht herzlich danken möchte ich an dieser Stelle Herrn Dr. Rüdiger Esser für die hervorragende Organisation dieses Gaststudentenprogramms, Sonja Dominiczak und Wolfgang Frings für die gute Betreuung meiner Arbeit. Außerdem danke ich Andreas Ernst für die vielen interessanten Unterhaltungen und die nette Atmosphäre im Büro und allen Gaststudenten und Mitarbeitern des ZAM für eine Zeit, an die ich noch lange und gerne denken werde

Literatur

1. Arbeitsgruppe Geodynamik, Institut für Geophysik der Westfälischen Wilhelms-Universität Münster, *http://earth.uni-muenster.de/dyn*
2. **Dominiczak, S.**, *Entwicklung einer Online-Visualisierung zu dem Simulationsprogramm NBODY6++ mit Hilfe der Kopplungsbibliothek VISIT*, Interner Bericht FZJ-ZAM-IB-2005-02, Januar 2005, 62 Seiten
3. **Elbeshausen, D.**, (in Vorbereitung:) *Diplomarbeit Geophysik*, Univ. Münster (vorr. Januar 2006)
4. **Frings, W.**, *Strategien zur Kopplung und Datenreduktion bei der Online-Visualisierung von parallelen Simulationsrechnungen mit verteilter Datenhaltung*, Forschungszentrum Jülich, Bericht Jül-4021, 2002
5. **Frings, W. und Eickermann, T.**, *VISIT: Ein Tool zur Online-Visualisierung und Steuerung von parallelen Simulationsrechnungen*, Mitteilungen - Gesellschaft für Informatik e. V., Parallel-Algorithmen und Rechnerstrukturen, 2003

6. **Frings, W. und Eickermann T.**, *VISIT - a Visualization Interface Toolkit - Version 1.0*, Forschungszentrum Jülich, ZAM - Technical Report IB-2000-16, 2000
7. **Harder, H. und Hansen, U.**, *A finite-volume solution method for thermal convection and dynamo problems in spherical shells*, Geophys. Journal Int., 161, 2005, pp. 522-532
8. JUelichMultiProcessor, Forschungszentrum Jülich, *http://jumpdoc.fz-juelich.de*
9. Unicore, *http://www.fz-juelich.de/unicore*
10. Zentralinstitut für angewandte Mathematik, Forschungszentrum Jülich, *http://www.fz-juelich.de/zam*

Autokorrelationsanalyse symplektischer Integratoren in der Gitter-Quantenchromodynamik

Jens Grieger

Institut für Physik der Humboldt Universität zu Berlin

E-mail: grieger@physik.hu-berlin.de

Zusammenfassung:

In dieser Arbeit werden zwei symplektische Integratoren für den in der Gitter-Quantenchromodynamik verwendeten Hybrid-Monte-Carlo-Algorithmus verglichen. Die beiden Integratoren sind der Minimum-Norm-Integrator 2. Ordnung (2MN) nach Omelyan et al. und der Leap-Frog-Integrator 2. Ordnung (2LF). Die Energiedifferenz $\langle \Delta H^2 \rangle^{1/2}$, d.h. der Fehler, der bei der Integration gemacht wird, ist mit dem 2MN etwa zehnmal kleiner als der Fehler bei der Integration mit dem 2LF [2]. Man erwartet somit, dass bei gleichem Fehler die Schrittweite $\Delta\tau$ bei der Integration mit dem 2MN um etwa einen Faktor 3 größer gewählt werden kann als mit dem 2LF. Es wird der potentielle Geschwindigkeitsvorteil beim Erzeugen von dynamischen Eichfeldkonfigurationen untersucht und dabei die Autokorrelation berücksichtigt, um eine Aussage über die effektiv erzeugten, d.h. statistisch unabhängigen Konfigurationen zu machen. Die Simulationen wurden mit dem frei verfügbaren Code der MILC Kollaboration durchgeführt [6]. Auf Grundlage der durchgeführten Simulationen kann der theoretische Geschwindigkeitsvorteil des 2MN gegenüber der Simulation mit dem 2LF nicht bestätigt werden. Ein signifikanter Unterschied in den Zahlen statistisch unabhängiger Konfigurationen, die mit den beiden Integratoren in gleicher Zeit erzeugt wurden, lässt sich nicht nachweisen.

Einleitung

Die Simulation dynamischer Quarks in der Gitter-Quantenchromodynamik (QCD) ist enorm zeitaufwendig. Dies gilt insbesondere für den physikalisch relevanten Fall kleiner Quarkmassen. Jede Möglichkeit, die Geschwindigkeit der Simulation zu erhöhen, ist daher von großem Interesse. Der Hybrid Monte Carlo Algorithmus (HMC) [1] ist heute der Standardalgorithmus zur Simulation dynamischer Fermionen in der Gitter-QCD. Er kombiniert die Methode der Molekulardynamik (MD) mit einem Metropolis-Test am Ende einer jeden MD-Trajektorie. In einer MD-Trajektorie werden Hamiltonsche Bewegungsgleichungen über eine fiktive Zeit $\tau = n\Delta\tau$ integriert. Dazu wird ein Schema benutzt, das die Integration in n Schritten durchführt und in jedem Schritt über die Zeit $\Delta\tau$ integriert. Der meist verwendete Integrator ist der Leap-Frog-Integrator 2. Ordnung (2LF), dessen Fehler $O(\Delta\tau^2)$ ist. Der Metropolis-Test sorgt in einem „accept/reject“ Schritt dafür, dass die gewünschte Wahrscheinlichkeitsverteilung erzeugt wird. Die Akzeptanzrate hängt somit von dem Fehler der Gesamtenergie nach einer Trajektorie ab.

Omelyan et al. schlagen mit dem Minimum-Norm-Integrator (2MN) einen neuen symplektischen Integrator zweiter Ordnung vor, dessen Fehler bei gleicher Schrittweite $\Delta\tau$ um einen Faktor 10 kleiner ist als der des 2LF [3]. Bei einem Integrator 2. Ordnung bedeutet dies, dass der Fehler des 2MN mit 3 mal größerer Schrittweite gleich dem Fehler des 2LF sein sollte. Dadurch könnte ein Geschwindigkeitsvorteil bei der aufwendigen Simulation dynamischer Fermionen erreicht werden.

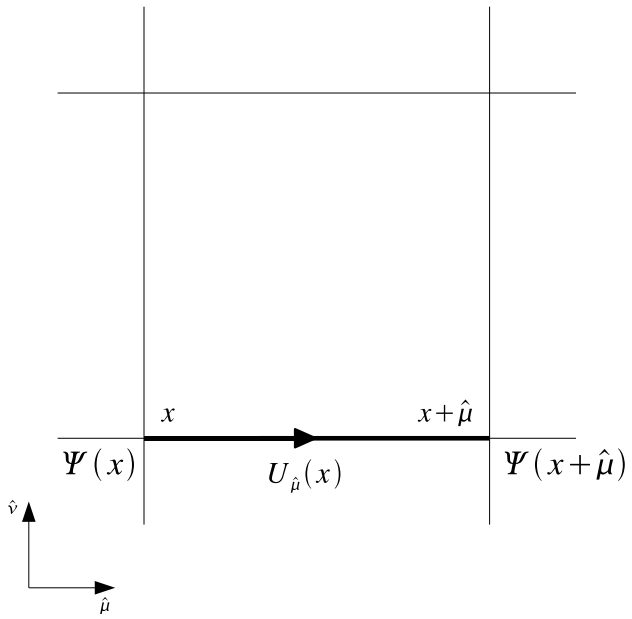


Abbildung 1: 2-dimensionale Ebene des Gitters.

Grundlagen

In der Natur sind vier fundamentale Wechselwirkungen bekannt: die Gravitation, die elektromagnetische, die schwache und die starke Wechselwirkung. Die Theorie der starken Wechselwirkung ist die Quanten-Chromodynamik. Die Auswertung dieser Theorie ist störungstheoretisch nur im Bereich hoher Energien möglich. Die fundamentalen Materieteilchen sind in der Quantenchromodynamik die Quarks. Diese sind Spin 1/2-Teilchen und somit Fermionen. Die Wechselwirkung zwischen den Quarks wird durch Eichbosonen, die sogenannten Gluonen vermittelt. Als Bindungszustände der Quarks sind Mesonen (z.B. das π-Meson) und Baryonen (z.B. das Proton und das Neutron) bekannt. In den Energiebereichen, in denen diese Bindungszustände vorliegen, sind störungstheoretische Rechnungen nicht mehr durchführbar. Aus diesem Grund wird die Theorie diskretisiert und auf einem 4-dimensionalen Raum-Zeit-Gitter ausgewertet.

In Abb. 1 ist eine 2-dimensionale Ebene des Gitters dargestellt. In einer Feldtheorie werden Teilchen als Felder aufgefasst. Die Materiefelder sind auf den Gitterpunkten definiert, zwischen den Punkten die Linkvariablen U, die Ausdruck für die Wechselwirkung zwischen den Teilchen sind. Die Linkvariablen sind SU(3) Matrizen. Die Gesamtheit dieser Linkvariablen nennt man eine Konfiguration $\{U_\nu\}$.

Alle weiteren Betrachtungen erfolgen in euklidischer Zeit t_E. Dazu wird eine sogenannte Wick-Rotation durchgeführt, die durch eine Rotation der Zeitkoordinate t auf die imaginäre Achse definiert ist:

$$t_E \rightarrow -it. \tag{1}$$

Ziel der Gitter-Simulation ist es, Erwartungswerte $\langle O \rangle$ von Observablen O zu bestimmen. Dazu muss ein Pfadintegral

$$\langle O \rangle = \frac{1}{Z} \int DUO[U] \det M[U] e^{-S_g[U]}, \tag{2}$$

mit der Normierung

$$Z = \int DU \det M[U] e^{-S_g[U]}, \tag{3}$$

berechnet werden. In diesem Funktionalintegral wird über alle möglichen Konfigurationen integriert. In der Gitterdiskretisierung gilt:

$$DU = \prod_{x,\mu} dU_\mu(x). \tag{4}$$

In (2) ist M die fermionische Wechselwirkungsmatrix der Wilson Fermionen,

$$M(x,y) = \delta_{xy} - \kappa \sum_{\mu} \left[(1-\gamma_\mu) U_\mu(x) \delta_{x+a\hat{\mu},x} (1+\gamma_\mu) U_\mu^\dagger(x-a\hat{\mu}) \delta_{x-a\hat{\mu},x} \right]. \tag{5}$$

Die Masse der Quarks m_q findet sich im „hopping parameter“ $\kappa = (2am_q + 8)^{-1}$ wieder, wobei a der Gitterabstand ist.

S_g ist die Wirkung des Eichfeldes, die sich wie folgt darstellen lässt:

$$S_g = \beta \sum_x \sum_{\mu<\nu} \left[\frac{1}{3} \text{Re Tr } W_{\mu\nu}^{1\times 1}(x) \right], \tag{6}$$

wobei β der Kopplungsparameter für das Eichfeld und $W_{\mu\nu}^{1\times 1}(x)$ der einfachste Wilson-Loop, die sogenannte Plaquette, ist. Die Wirkung (6) wird auch Plaquette-Wirkung genannt.

Stellt man in (2) die Determinante der Fermionmatrix mit Hilfe von Pseudo-Fermionenfeldern Φ bzw. $\Phi^\dagger$ dar, so ergibt sich mit der effektiven Wirkung $S_{\text{eff}}[U, \Phi^\dagger, \Phi]$, bestehend aus der Wirkung des Eichfeldes $S_g[U]$ und der Wirkung der Pseudofermionen $S_{\text{pf}}[U, \Phi^\dagger, \Phi]$,

$$S_{\text{eff}}[U, \Phi^\dagger, \Phi] = S_g[U] + S_{\text{pf}}[U, \Phi^\dagger, \Phi], \tag{7}$$

folgender Ausdruck für den Erwartungswert (2):

$$\langle O \rangle = \frac{1}{Z} \int D\Phi^\dagger D\Phi DU O[U] e^{-S_{\text{eff}}[\Pi, U, \Phi^\dagger, \Phi]}, \tag{8}$$

mit

$$Z = \int D\Phi^\dagger D\Phi DU e^{-S_{\text{eff}}[\Pi, U, \Phi^\dagger, \Phi]}. \tag{9}$$

Monte-Carlo-Simulation

Das Ziel einer Simulation in der Gitter-QCD besteht letztendlich darin, gemäß (8) den Erwartungswert einer Observablen zu berechnen. Um (8) auszuwerten, muss im Prinzip über alle denkbaren Konfigurationen integriert werden, was in der Praxis aber nicht möglich ist. Im Rahmen einer Monte-Carlo-Simulation wird versucht, ein repräsentatives Ensemble an Konfigurationen zu erzeugen. Diese Konfigurationen sollen der dem Gewichtsfaktor $\exp[-S_{\text{eff}}[U]]$ im Pfadintegral entsprechenden Wahrscheinlichkeitsverteilung genügen. Das Erzeugen eines Ensembles mit bestimmter Wahrscheinlichkeitsverteilung nennt man „importance sampling“. Es hat den Vorteil, dass sich sich der Erwartungswert einfach durch folgende Summe darstellen lässt:

$$\langle O \rangle = \frac{1}{N} \sum_{i=1}^{N} O[U_i]. \tag{10}$$

Markov Prozess

Bei einem Markov-Prozess wird aus einer vorhandenen Konfiguration eine neue erzeugt:

$$\{U_1\} \overset{P(\{U_1\}\to\{U_2\})}{\longrightarrow} \{U_2\} \overset{P(\{U_2\}\to\{U_3\})}{\longrightarrow} \{U_3\},$$

mit $P(\{U_1\} \to \{U_2\})$ als Übergangswahrscheinlichkeit von $\{U_1\}$ nach $\{U_2\}$.

Hierdurch wird deutlich, dass erzeugte Konfigurationen nicht unabhängig voneinander sein können. Das Thema Autokorrelation wird im nächsten Kapitel behandelt.

Das Problem der Konfigurationserzeugung reduziert sich nun auf das Finden der richtigen Übergangswahrscheinlichkeit. Durch eine Markov-Kette sind Übergangswahrscheinlichkeiten $P(\{U_1\} \to \{U_2\})$ bzw. $P_{\mu\nu}$ gegeben, die ein Ensemble mit gewünschter Wahrscheinlichkeitsverteilung p_μ erzeugen. Um zu gewährleisten, dass durch diesen Prozess die richtige Verteilung erzeugt wird, muss die Markov-Kette folgende Eigenschaften haben:

$$\sum_\nu P_{\mu\nu} = 1, \tag{11}$$

$$\forall(\mu,\nu) : \{U_\mu\} \to \{U_\nu\} \text{ möglich in endlich vielen Schritten.} \tag{12}$$

Bedingung (11) ist eine Normierung. Die Bedingung (12) der Ergodizität besagt, dass es möglich ist, in endlich vielen Schritten von jeder Konfiguration $\{U_\mu\}$ zur Konfiguration $\{U_\nu\}$ zu kommen.

Eine hinreichende, jedoch nicht notwendige Bedingung, (11) und (12) zu erfüllen, ist die Eigenschaft der „detailed balance“:

$$p_\mu P_{\mu\nu} = p_\nu P_{\nu\mu}. \tag{13}$$

Der Metropolis-Algorithmus [5] genügt der Forderung (13)

$$P_{\mu\nu} = \min\left(1, \frac{p_\nu}{p_\mu}\right) = \min\left(1, \frac{e^{-S[U_\nu]}}{e^{-S[U_\mu]}}\right) = \min\left(1, e^{-\Delta S}\right), \tag{14}$$

wobei $\Delta S = S[U_\nu] - S[U_\mu]$ die Differenz der Wirkungen mit der neuen und der alten Konfiguration ist.

Um eine neue Konfiguration zu erhalten wird zunächst ein Vorschlag gemacht und dann nach Metropolis getestet, ob die neue Konfiguration der geforderten Wahrscheinlichkeitsverteilung genügt.

Hybrid Monte Carlo

Wird die Wirkung mit einer neuen Konfiguration größer und weicht stark von der alten ab, ist die Wahrscheinlichkeit, den Metropolis-Test zu bestehen, sehr gering. Bei einem nicht akzeptierten Vorschlag geht die alte Konfiguration erneut in das Ensemble ein. Eine Methode, mit der eine hohe Akzeptanzrate realisiert werden kann, ist der Hybrid-Monte-Carlo Algorithmus. Dabei wird eine Molekulardynamik-Simulation gefolgt von einem Metropolis-Test gemäß (14) durchgeführt. Bei einer Molekulardynamik-Simulation werden die Hamiltonschen Bewegungsgleichungen iterativ in n Schritten gelöst, wobei in jedem Schritt über die Zeit $\Delta\tau$ integriert wird. Die Integration über die fiktive Zeit $\tau = n\Delta\tau$ und die daraus resultierende Eichfeldkonfiguration nennt man auch Trajektorie.

Die Gesamtenergie des dynamischen Systems wird beschrieben durch den Hamiltonian

$$H[\Pi, U, \Phi^\dagger, \Phi] = T[\Pi] + S_{\text{eff}}[U, \Phi^\dagger, \Phi], \tag{15}$$

mit

$$T = \frac{1}{2}\sum_{x,\mu} \text{Tr}\, \Pi_\mu^2(x). \tag{16}$$

Π ist der kanonisch konjugierte Impuls zu U.

Der Erwartungswert (8) kann auch mit dem Hamiltonian berechnet werden:

$$\langle O\rangle = \frac{1}{Z}\int D\Phi^\dagger D\Phi DU O[U] e^{-H[\Pi,U,\Phi^\dagger,\Phi]}, \tag{17}$$

mit

$$Z = \int D\Phi^\dagger D\Phi DU e^{-H[\Pi, U, \Phi^\dagger, \Phi]}. \tag{18}$$

(17) ist equivalent zu (8), da die Konstante, die sich aus der Integration der konjugierten Impulse ergibt, durch die Normierung aufgehoben wird.

Man gelangt nun zu einer neuen Konfiguration, indem man fordert, dass sich die Gesamtenergie des Systems nicht ändert, d.h. $\dot{H}[\Pi, U, \Phi^\dagger, \Phi] = 0$

Aus dieser Forderung und den beiden Hamiltonschen Bewegungsgleichungen

$$\dot{U}_\mu = \frac{\partial H}{\partial \Pi_\mu} \tag{19}$$

und

$$\dot{\Pi}_\mu = -\frac{\partial H}{\partial U_\mu} \tag{20}$$

ergibt sich ein System von zwei gekoppelten Differentialgleichungen:

$$\dot{U}_\mu = i\Pi_\mu U_\mu, \tag{21}$$

$$\dot{\Pi}_\mu = -iF_\mu. \tag{22}$$

Ein symplektischer Integrator löst dieses System gekoppelter Differentialgleichungen bei Erhaltung der Energie. In (22) ist $F[U]$ die fermionische Kraft. Um diese zu berechnen muss die fermionische Matrix invertiert werden, was im HMC die meiste Rechenzeit in Anspruch nimmt.

Für einen kleinen Zeitschritt $\Delta\tau$ gelten folgende Entwicklungen:

$$\Pi(\tau + \Delta\tau) = \Pi(\tau) - i\Delta\tau F[U(\tau)], \tag{23}$$

$$U(\tau + \Delta\tau) = \exp[i\Delta\tau\Pi(\tau)]\, U(\tau). \tag{24}$$

Symplektischer Integrator

Die numerische Lösung des System der gekoppelten Differentialgleichungen verletzt die Forderung nach Erhaltung der Gesamtenergie. Die Größe des Fehlers wird deutlich, wenn man die Hamiltonschen Bewegungsgleichungen (19) und (20) mit Hilfe der Poisson-Klammer darstellt und auswertet:

$$\dot{f} = \{f, H\} = \sum_i \left(\frac{\partial f}{\partial U}\frac{\partial H}{\partial \Pi} - \frac{\partial f}{\partial \Pi}\frac{\partial H}{\partial U} \right) \tag{25}$$

Für f kann man U bzw. Π einsetzen und erhält so (19) bzw. (20).

Definiert man den linearen Operator $L(H)$ als

$$L(H)f \equiv \{f, H\}, \tag{26}$$

so lassen sich die Bewegungsgleichungen formal lösen:

$$f(\tau + \Delta\tau) = \exp(\Delta\tau L(H)) f(\tau), \tag{27}$$

wobei nach (15) der Operator $\exp(\Delta\tau L(H))$ auch wie folgt ausgedrückt werden kann:

$$\exp[\Delta\tau L(H)] = \exp[\Delta\tau(L(T) + L(S))]. \tag{28}$$

Zur einfacheren Darstellung werden $\tilde{T} \equiv L(T)$ und $\tilde{V} \equiv L(S)$ definiert.

Bei der numerischen Berechnung lässt sich der Operator $\exp(\Delta\tau L(H))$ nicht exakt auf $f(\tau)$ anwenden. Um (27) zu lösen, wird der Operator approximiert und je nach Darstellung ein Fehler bestimmter Ordnung von $\Delta\tau$ gemacht. In den nächsten beiden Unterkapiteln werden die beiden Integratoren vorgestellt, bei denen durch eine unterschiedliche Darstellung des Operators (28) Fehler unterschiedlicher Größe gemacht werden.

Leap-Frog Integration

Beim Leap-Frog Integrationsschema wird der Operator wie folgt dargestellt:

$$e^{\Delta\tau(\tilde{T}+\tilde{V})} = e^{\frac{1}{2}\Delta\tau\tilde{T}} e^{\Delta\tau\tilde{V}} e^{\frac{1}{2}\Delta\tau\tilde{T}} + O(\Delta\tau^3). \tag{29}$$

Für eine zeitliche Entwicklung der Felder U und Π von τ nach $\tau + \Delta\tau$, d.h. für Integration über ein Teilintervall $\Delta\tau$, werden folgende Teilschritte gemacht:

$$\begin{array}{lll} U(0) & \xrightarrow{\frac{\Delta\tau}{2}} & U\left(\frac{\Delta\tau}{2}\right) = \exp\left[i\frac{\Delta\tau}{2}\Pi(0)\right] U(0), \\ \Pi(0) & \xrightarrow{\Delta\tau} & \Pi(\Delta\tau) = \Pi(0) - i\Delta\tau F\left[U\left(\frac{\Delta\tau}{2}\right)\right], \\ U\left(\frac{\Delta\tau}{2}\right) & \xrightarrow{\frac{\Delta\tau}{2}} & U(\Delta\tau) = \exp\left[i\frac{\Delta\tau}{2}\Pi(\Delta\tau)\right] U\left(\frac{\Delta\tau}{2}\right). \end{array}$$

Bei einem Intgrator 2. Ordnung hat der Fehler für ein Teilintervall die Struktur:

$$O(\Delta t^3) = \alpha[\tilde{T}, [\tilde{V}, \tilde{T}]] + \beta[\tilde{V}, [\tilde{V}, \tilde{T}]], \tag{30}$$

wobei sich die Größe des Fehlers wie folgt darstellen lässt:

$$Err \equiv \sqrt{\alpha^2 + \beta^2} \tag{31}$$

Minimum-Norm Integration

Der Minimum-Norm-Integrator ist ebenfalls 2. Ordnung und somit hat der Fehler ebenfalls die Struktur von (30) bzw. (31). Die Idee bei der Konstruktion dieses Schemas ist, α und β so zu wählen, dass der Fehler Err minimal wird. Die Darstellung des Operators (28) ergibt sich dann zu:

$$e^{\Delta\tau(\tilde{T}+\tilde{V})} = e^{\lambda\Delta\tau\tilde{T}} e^{\frac{1}{2}\Delta\tau\tilde{V}} e^{(1-2\lambda)\Delta\tau\tilde{T}} e^{\frac{1}{2}\Delta\tau\tilde{V}} e^{\lambda\Delta\tau\tilde{T}} + O(\Delta\tau^3). \tag{32}$$

Für eine zeitliche Entwicklung der Felder U und Π von τ nach $\tau + \Delta\tau$, d.h. für Integration über ein Teilintervall $\Delta\tau$, werden hier fünf Teilschritte gemacht:

$$\begin{array}{lll} U(0) & \xrightarrow{\lambda\Delta\tau} & U(\lambda\Delta\tau) = \exp\left[i\lambda\Delta\tau\Pi(0)\right] U(0), \\ \Pi(0) & \xrightarrow{\frac{\Delta\tau}{2}} & \Pi\left(\frac{\Delta\tau}{2}\right) = \Pi(0) - i\frac{\Delta\tau}{2} F\left[U(\lambda\Delta\tau)\right], \\ U(\lambda\Delta\tau) & \xrightarrow{(1-2\lambda)\Delta\tau} & U((1-\lambda)\Delta\tau) = \exp\left[i(1-2\lambda)\Delta\tau\Pi\left(\frac{\Delta\tau}{2}\right)\right] U(\lambda\Delta\tau), \\ \Pi\left(\frac{\Delta\tau}{2}\right) & \xrightarrow{\frac{\Delta\tau}{2}} & \Pi(\Delta\tau) = \Pi\left(\frac{\Delta\tau}{2}\right) - i\frac{\Delta\tau}{2} F\left[U((1-\lambda)\Delta\tau)\right], \\ U((1-\lambda)\Delta\tau) & \xrightarrow{\lambda\Delta\tau} & U(\Delta\tau) = \exp\left[i\lambda\Delta\tau\Pi(\Delta\tau)\right] U((1-\lambda)\Delta\tau). \end{array}$$

Der Wert für λ lässt sich aus der Forderung an einen minimalen Fehler Err errechnen [2]:

$$\lambda \approx 0.1931833275037836. \tag{33}$$

Ein theoretischer Vergleich der Fehler der beiden Integratoren ergibt, dass der Fehler des Minimum-Norm-Integrators um einen Faktor ≈ 10 geringer ist:

$$\frac{Err^{lf}}{Err^{mn}} \approx 10.9. \tag{34}$$

Fehler des Hamiltonians nach einer Trajektorie

Um eine neue Konfiguration zu erzeugen wird die Integration eines Teilintervalls n-mal wiederholt, d.h. die beiden Felder U und Π sind am Ende einer Trajektorie am Zeitpunkt $\tau = n\Delta\tau$. Da bei jedem einzelnen Integrationsschritt ein Fehler gemacht wird, steigt der Gesamtfehler mit der Zahl der Schritte an, jedoch nur bis zu einer charakteristischen Trajektorienlänge l_c. Ab dieser Länge wird der Fehler nicht mehr größer. In (30) wurde der Fehler der Integration dargestellt. Für die Differenz der Gesamtenergien vor und nach einer Trajektorie, ΔH, gilt [7]

$$\Delta H \sim O(\Delta t^3)\frac{l_c}{\Delta t}, \tag{35}$$

so dass der Gesamtfehler $O(\Delta t^2)$ ist.

Theoretischer Vergleich der Leap-Frog- und Minimum-Norm-Integration

Bei einem Update des kanonisch konjugierten Impulsfeldes Π muss zur Berechnung der Kraft $F[U]$ die fermionische Matrix invertiert werden. Dieser Schritt nimmt im HMC die meiste Zeit in Anspruch. Die Berechnung der Kraft wird in einem Zeitschritt $\Delta\tau$ bei der Leap-Frog-Integration nur einmal, bei der Minimum-Norm-Integration hingegen zweimal durchgeführt.

Der Fehler beider Integratoren hängt quadratisch von der Schrittweite ab. Nach (34) sollte somit der Fehler bei der Integration mit dem 2MN gleich dem Fehler bei der Integration mit dem 2LF sein, wenn die Schrittweite $\Delta\tau$ bei der Minimum-Norm-Integration um den Faktor $\sqrt{10.9}$ vergrößert wird.

Werden jetzt bei der Minimum-Norm-Integration entsprechend weniger Schritte n gemacht, wäre zu erwarten, dass diese um den Faktor $\sqrt{10.9}/2$ schneller ist, da bei der Minimum-Norm-Integration die Berechnung der Kraft $F[U]$ zweimal durchgeführt wird, jedoch nur $\approx 1/3$ der Schritte gemacht werden müssen, um den gleichen Fehler zu erhalten.

Autokorrelation

Es wurde schon angesprochen, dass aufeinanderfolgende Konfigurationen im Allgemeinen nicht unabhängig voneinander sind. Bei der Fehlerbetrachtung muss dies berücksichtigt werden. Nach (10) kann der gewünschter Erwartungswert einer Observablen O aus dem arithmetischen Mittel über die Markov-Kette bestimmt werden:

$$\bar{O} = \frac{1}{N}\sum_{i=1}^{N} O_i. \tag{36}$$

Hierbei ist $\bar{O}$ vom Erwartungswert $\langle O\rangle$ zu unterscheiden. $\langle O\rangle$ ist der theoretische Erwartungswert, $\bar{O}$ eine zufällig um den Erwartungswert schwankende Größe. Der Fehler bzw. die Varianz lässt sich wie folgt abschätzen:

$$\sigma^2_{\bar{O}} = \langle\left[\bar{O} - \langle\bar{O}\rangle\right]\rangle = \langle\bar{O}^2\rangle - \langle\bar{O}\rangle^2. \tag{37}$$

Wenn alle O_i unabhängig sind, lässt sich die Varianz einfach abschätzen:

$$\sigma^2_{\bar{O}} = \sigma^2_{O_i}/N, \tag{38}$$

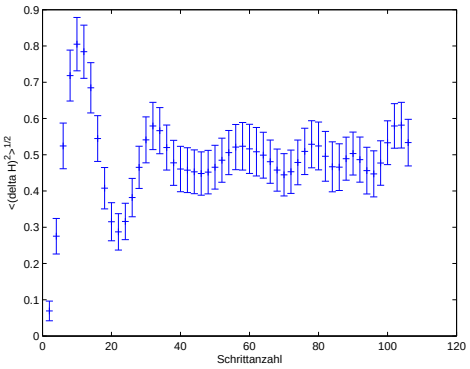


Abbildung 2: 2LF, $\Delta\tau = 1/27$

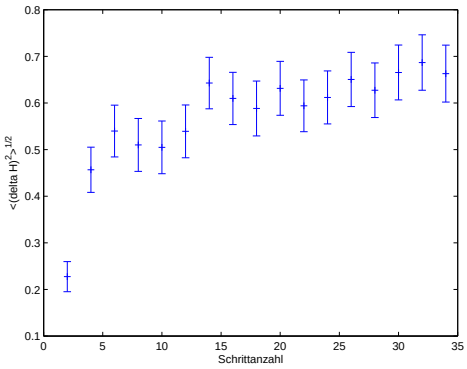


Abbildung 3: 2MN, $\Delta\tau = 1/9$

wobei $\sigma^2_{O_i} = \langle O_i^2 \rangle - \langle O_i \rangle^2$ ist.

Da dies jedoch nicht gegeben ist, wird der Fehler im Allgemeinen größer werden:

$$\sigma^2_{\bar{O}} = \frac{\sigma^2_{O_i}}{N} 2\tau_{O,\text{int}}. \tag{39}$$

Mit Einführung der integrierten Autokorrelationszeit $\tau_{O,\text{int}}$ lässt sich eine effektive Anzahl von Meßwerten N_{eff}, d.h. eine Anzahl statistisch unabhängiger Konfigurationen definieren:

$$N_{\text{eff}} = \frac{N}{2\tau_{O,\text{int}}}. \tag{40}$$

Simulation und Ergebnisse

Gerechnet wurde auf „JUMP“ (JUelich Multi-Processor), dem IBM p690 Cluster des John von Neumann Instituts für Computing (NIC) in Jülich. Die Simulationen wurden mit dem frei erhältlichen Code der MILC-Kollaboration in der Version 6 vom 20. September 2002 durchgeführt. Es wurde auf einem Gitter der Größe $8^3 \times 16$ mit den Parametern $\beta = 5.6$ und $\kappa = 0.145$ simuliert. Zur Auswertung der Autokorrelation wurde die in [4] beschriebene Methode mittels der vom Autor zur Verfügung gestellten MATLAB-Routine verwendet.

Zunächst ist es wichtig, geeignete Parameter für die Schrittweite $\Delta\tau$ und Schrittanzahl n der Teilintervalle für die Integration zu finden. Wie in [2] vorgeschlagen wurde die Akzeptanzrate auf $\approx 60\%$ eingestellt. Um die Sättigung des Fehlers der Integratoren nach (35) zu überprüfen wurde bei gleicher Schrittweite $\Delta\tau$ die Schrittanzahl n variiert (Abb. 2 und Abb. 3). Mit den Schrittweiten $\Delta\tau = 1/27$ für die Leap-Frog-Integration (Abb. 2) und $\Delta\tau = 1/9$ für die Minimum-Norm-Integration (Abb. 3) stellte sich im Bereich der Sättigung die Akzeptanzrate von $\approx 60\%$ ein.

Die in Tabelle 1 aufgeführte Simulation mit der Leap-Frog-Integration wurde mit $n = 54$ Schritten durchgeführt. Bei der Minimum-Norm-Integration betrug die Zahl der Schritte $n = 18$. Somit ergibt sich für beide Integratoren eine gleiche Trajektorienlänge von $\tau = n\Delta\tau = 2$.

Vergleicht man die Zahl der während einer Laufzeit erzeugten Konfigurationen, so ist zu berücksichtigen, dass bei dem Programmdurchlauf Berechnungen (z.B. des Hadron-Spektrums) gemacht werden, die nicht direkt der Konfigurationserzeugung dienen. Somit kann die absolute Zahl erzeugter Konfigurationen in gleicher Laufzeit auch nicht direkt verglichen werden. Aussagekräftiger ist, es nur die mittlere Zeit t_{update} zu vergleichen, die unmittelbar zum Erzeugen einer Konfiguration aufgewendet wurde.

	leapfrog	minimum norm
Simulationsdauer	12 h	12 h
Schrittweite	$\Delta\tau = 1/27$	$\Delta\tau = 1/9$
Schrittanzahl	$n = 54$	$n = 18$
erzeugte Konfigurationen	6453	7877
Akzeptanzrate	$\approx 63.7\%$	$\approx 59.4\%$
$\langle \Delta H^2 \rangle^{1/2}$	1.06(1)	1.23(2)
Plaquette	1.6624(4)	1.6628(4)
τ_{int}	7(1)	10(2)
t_{update}	3.9205(1) s	2.8639(1) s

Tabelle 1: Simulation auf einem $8^3 \times 16$ Gitter mit $\beta = 5.6$ und $\kappa = 0.145$.

Bei der Erzeugung einer neuen Konfiguration dauert die Berechnung des Π Feldes am längsten, da hierfür die fermionische Kraft berechnet und somit die fermionische Matrix invertiert werden muss. Diese Berechnung wird beim 2LF nur einmal, beim 2MN dagegen zweimal durchgeführt. Da bei der Simulation die Schrittweite des Minimum-Norm-Integrators um einen Faktor 3 größer gewählt wird, ist ein theoretisches Verhältnis $t_{\text{update}}^{\text{lf, theo}} / t_{\text{update}}^{\text{mn, theo}} \approx 1.5$ zu erwarten.

Gemessen wurde jedoch folgendes Verhältnis:

$$\frac{t_{\text{update}}^{\text{lf}}}{t_{\text{update}}^{\text{mn}}} = 1.3689(1) \tag{41}$$

Nimmt man den Fall an, dass das „Update“ der Konfigurationen die gesamte Simulationsdauer einnimmt, so würde in der Simulationszeit T eine Anzahl von $N = T/t_{\text{update}}$ Konfigurationen erzeugt werden.

Nach (40) ergibt sich für das Verhältnis der Zahlen effektiv erzeugter Konfigurationen mit dem Minimum Norm Integrator, $N_{\text{eff}}^{\text{mn}}$, und dem Leap Frog Integrator, $N_{\text{eff}}^{\text{lf}}$:

$$\frac{N_{\text{eff}}^{\text{mn}}}{N_{\text{eff}}^{\text{lf}}} = \frac{\tau_{\text{int}}^{\text{lf}} t_{\text{update}}^{\text{lf}}}{\tau_{\text{int}}^{\text{mn}} t_{\text{update}}^{\text{mn}}} = 1.0(3). \tag{42}$$

Im Rahmen des Fehlers konnte nach (42) bei der Erzeugung statistisch unabhängiger Konfigurationen in gleicher Simulationszeit kein Vorteil des 2MN gegenüber dem 2LF festgestellt werden. Desweiteren ist zu beachten, dass das Verhältnis der mittleren Update-Zeiten (41) schlechter ist als theoretisch erwartet. Dies lässt sich auf eine unterschiedliche Dauer der Inversion der fermionischen Matrix zurückführen. Die Inversion wird mit der Methode der Konjugierten Gradienten durchgeführt, bei der ausgehend von einem Startvektor die Lösung eines linearen Gleichungssystems iterativ berechnet wird. Um dieses Verfahren zu beschleunigen, wird der Startvektor jeweils aus den beiden vorangegangenen Lösungsvektoren extrapoliert.

Bei der kleinen Schrittweite $\Delta\tau = 1/27$, die für die Leap-Frog-Integration verwendet wurde, brachte das einen Vorteil. Es mussten weniger Iterationen durchgeführt werden, um sich der Lösung mit der gewünschten Genauigkeit anzunähern. Die gewählte Schrittweite $\Delta\tau = 1/9$ bei der Minimum-Norm-Integration war um einen Faktor 3 größer. Bei dieser Schrittweite brachte die Extrapolation des Startvektors aus den beiden vorherigen Lösungen keinen Vorteil.

Tabelle 2 zeigt einen Vergleich der mittleren Anzahl von Iterationsschritten mit festem Startvektor und mit einem Startvektor, der vor jedem Iterationsschritt neu extrapoliert wurde.

	leapfrog $\Delta\tau = 1/27$	minimum norm $\Delta\tau = 1/9$
Anzahl Iterationen mit festem Startvektor	$34.4(2)$	$37.0(1)$
Anzahl Iterationen mit extrapoliertem Startvektor	$29.8(1)$	$37.1(1)$

Tabelle 2: Vergleich der Anzahl an Iterationen mit und ohne extrapoliertem Startvektor.

Zusammenfassung

Theoretisch sollte der Fehler des Minimum-Norm-Integrators bei gleicher Schrittweite $\Delta\tau$ um einen Faktor 10.9 kleiner als der Fehler des Leap-Frog-Integrators sein. Da der Fehler quadratisch von der Schrittweite $\Delta\tau$ abhängt, sollte der Fehler bei einer Vergrößerung der Schrittweite des Minimum-Norm-Integrators um den Faktor $\sqrt{10.9} \approx 3.3015$ gleich bleiben. In unserer Untersuchung war der Fehler bei der Erhöhung der Schrittweite um den Faktor 3 jedoch schon etwas größer.

Durch das Verhältnis $N_{\text{eff}}^{\text{mn}}/N_{\text{eff}}^{\text{lf}} = 1.0(3)$ der statistisch unabhängigen Konfigurationen konnte im Rahmen des Fehlers bei den durchgeführten Simulationen kein Vorteil des 2MN gegenüber dem 2LF festgestellt werden.

Das Verhältnis $t_{\text{update}}^{\text{lf}}/t_{\text{update}}^{\text{mn}} = 1.3689(1)$ war schlechter als erwartet. Dies lag daran, dass eine Vergrößerung der Schrittweite $\Delta\tau$, die eine Verringerung der Schrittanzahl n erlaubte, um die Felder $U(\tau)$ und $\Pi(\tau)$ zur gleichen Zeit $\tau = n\Delta\tau$ zu entwickeln, zwar bedeutet, dass die Häufigkeit der Berechnung der fermionischen Kraft $F[U]$ geringer war, diese jedoch länger dauerte.

Danksagung

Ich möchte hiermit meinem Betreuer Herrn Dr. Boris Orth danken. Außerdem danke ich Herrn Stefan Krieg. Desweiteren danke ich dem NIC und Herrn Dr. Rüdiger Esser für die Durchführung dieses Gaststudierendenprogramms. Ich möchte mich bei Herrn Prof. Ulli Wolff bedanken, der mir die Teilnahme an diesem Programm ermöglichte.

Außerdem danke ich allen anderen Gaststudierenden, insbesondere Aiko Voigt, Tobias Hertkorn und des weiteren Ross und Paddy Lynch.

Literatur

1. H.J. Rothe, Lattice Gauge Theories, World Scientific.
2. Tetsuya Takaishi, Philippe de Forcrand (2005) [arXiv:hep-lat/0505020].
3. I.P. Omelyan, I.M. Mryglod and R. Folk, Comput. Phys. Commun. **151** (2003) 272.
4. Ulli Wolff (2004) [arXiv:hep-lat/0306017].
5. Metropolis et al., Jour. Chem. Phys. 21(1953), 1087-1092.
6. MILC: http://www.physics.utah.edu/∼detar/milc/index.html
7. S. Gupta, A. Irback, F. Karsch and B. Petersson, Phys. Lett. B **242** (1990) 437.

Paralleles Rendering unter Verwendung eines Linux-Clusters

Britta Hennecken

Universität Koblenz

E-Mail: brittahe@uni-koblenz.de

Zusammenfassung:

Der Betrieb heutiger Graphikanwendungen übersteigt das Rechenvolumen handelsüblicher PCs und Workstations um ein Vielfaches. Aus diesem Grund wurden in der Vergangenheit spezielle Graphik-Rechner entwickelt, die die nötige Rechenleistung auf einer Maschine zur Verfügung stellen. Mehrere CPUs und GPUs (Graphics Processing Unit) sind in einem System vereint und arbeiten auf einem gemeinsamen Speicher. Der Betrieb rechenintensiver Graphikanwendungen kann mit Hilfe eines solchen Shared-Memory-Systems stabil und performant gewährleistet werden. Diese Graphikrechner bringen, neben ihren leistungsorientierten Vorzügen, gravierende ökonomische Nachteile mit sich. Die Anschaffung, Wartung und der Unterhalt einer solchen Maschine sind sehr kostenintensiv.

Eine geeignete Alternative bieten sogenannte Render-Cluster. Es handelt es sich um den Zusammenschluss leistungsstarker PCs oder Workstations zu einem Netzwerk, das Graphikanwendungen verteilt berechnet. Im Vergleich zum speziellen Graphikrechner ist dies eine kostengünstige und leistungsstarke Alternative zum Betrieb von Virtual Reality (VR)-Systemen. Da ein Cluster nahezu beliebig erweiterbar ist, kann sehr hohe Rechenleistung erzielt werden. Die Holobench des Forschungszentrums Jülich wird derzeit mit einer SGI-Onyx2 betrieben. Dieser Graphikrechner soll zukünftig durch ein Render-Cluster ersetzt werden. In dieser Ausarbeitung werden die Probleme und Möglichkeiten, die ein Render-Cluster mit sich bringt, betrachtet.

Einleitung

Das Hauptaugenmerk liegt darauf, eine Grahik-Anwendung nicht über ein Shared-Memory-System, sondern auf einem Netzwerk von eigenständigen Maschinen, zu betreiben. Es ist anzumerken, dass der Zugriff übers Netzwerk auf die einzelnen Komponenten um ein Vielfaches langsamer ist, als der Zugriff auf einen internen Speicher. Die einzelnen Graphik-Anweisungen eines Programms können in einem speziellen Graphik-Rechner hintereinander und zusammenhängend ausgeführt werden. Sie können auf einen gemeinsamen Speicher zugreifen und rechnen. Bei dem Betrieb der Anwendung über ein Render-Cluster muss dieser Anweisungs-Stream zerlegt werden. Die einzelnen Rechner, aus denen ein Cluster zusammengesetzt ist, werden Nodes genannt. Jedem Node wird eine Teilaufgabe zur Abarbeitung des gesamten Programms zugeteilt. Diese Zuordnung muss sinnvoll und effizient geschehen.

Des weiteren müssen die Teileinheiten synchronisiert und gesteuert werden. Auf den sogenannten Application-Nodes läuft das eigentliche Programm und der Anweisungs-Stream wird zerlegt. Die Nodes, die die eigentliche Ausgabe liefern, werden Render-Nodes genannt. Die Anordnung der Nodes innerhalb des Clusters, Topologie genannt, ist vom angewandten Verfahren abhängig.

Die Anwendung wird demnach über das Netzwerk verteilt, die einzelnen Berechnungen lokal auf den Nodes durchgeführt und wieder zu einem gemeinsamen Ergebnis zusammengefügt. Es gibt einige Tools

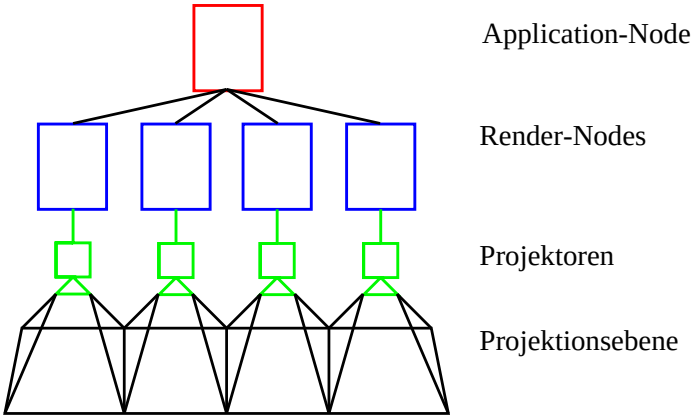


Abbildung 1: *Beispiel-Cluster zur Umsetzung des Sort-First-Verfahrens*

und Bibliotheken, die sich mit der Verteilung von Graphikanwendungen auf ein Render-Cluster beschäftigen und versuchen die aufgeführten Probleme zu beheben.

Im Folgenden werden Verfahren des parallelen Renderings unter Verwendung von PC-Clustern näher beleuchtet, das Tool Chromium auf seine Möglichkeiten untersucht und die Umstellung von der SGI-Onyx2 auf ein Linux-Cluster betrachtet.

Verfahren zum Parallelen Rendern

Im Allgemeinen verfolgt der Einsatz Parallelen Renderns zwei zentrale Ziele. Zum einen kann die Präsentation der Anwendung im Mittelpunkt stehen. Hier kommen Systeme wie Mehrseitenprojektionen oder Tiled-Displays zum Einsatz. Unter Mehrseitenprojektion versteht man die Darstellung auf einer nicht planaren Ausgabefläche, wie sie beispielsweise die Holobench oder CAVE darbieten. Tiled-Display-Systeme setzen die finale Ausgabe aus mehreren Teilbildern zusammen. Zum anderen kann aber auch die eigentliche Berechnungen sehr komplexer Anwendungen im Focus stehen. Die Präsentation ist hier zweitrangig. Um die verschiedenen Grundvoraussetzungen der Hardware mit denen der Software aufeinander abzustimmen, wurden verschiedene Verfahren zum Parallelen Rendering entwickelt. Namentlich orientieren sie sich an dem Zeitpunkt der Geometrie-Sortierung.

Sort-First-Verfahren

Bei diesem Verfahren wird die Ausgabe meist von mehreren Geräten auf eine oder mehrere Leinwände projiziert. Tiled-Display-Systeme lassen sich auf diese Weise effektiv realisieren. Ein mögliches Cluster zur Umsetzung dieses Verfahrens ist in Fig.1 dargestellt.

Der Application-Node, auf dem das Graphik-Programm läuft, steuert das gesamte Render-Cluster. Jeder Render-Node ist für die Darstellung einer bestimmten Bildkachel zuständig. Um dies unterstützen zu können muss die Geometrie zu Beginn der Bearbeitung sortiert werden. Die Render-Nodes erhalten nur für ihre Darstellung relevante Geometriedaten. Zur Zerlegung der Ausgabe in einzelne Teilbereiche können verschiedene Algorithmen herangezogen werden. Es gibt sowohl statische, als auch dynamische Verfahren. Die statischen weisen einem Render-Node feste Koordinaten zur Definition einer Bildkachel zu. Demnach berechnet ein Node, während des gesamten Programmablaufs, immer die selbe Bildeinheit (z.B. die obere, linke Ecke). Aufwändiger in Entwicklung und Betrieb sind dynamische Verfahren. Für jeden Frame wird die optimale Verteilung der Kacheln über die Ausgabe bestimmt. Auf diese Weise kann eine annähernd identische Last erreicht werden. Es ist sinnvoll, die Render-Nodes direkt an Ausgabegeräte, z.B. Projektoren, anzuschließen da somit Netzkommunikation gespart werden kann. Diese müssen

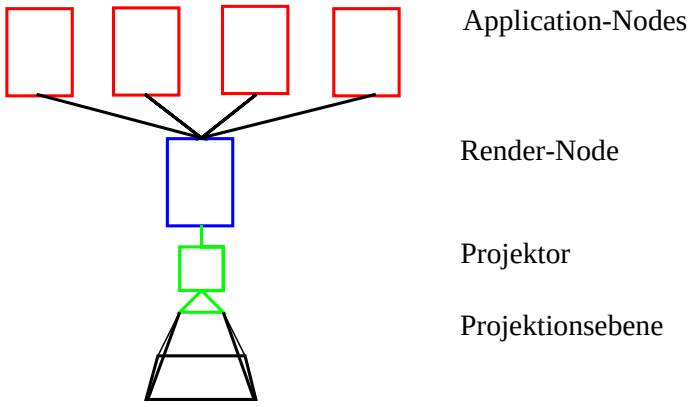


Abbildung 2: *Beispiel-Cluster zur Umsetzung des Sort-Last-Verfahrens*

exakt justiert sein, um ein harmonisches Gesamtbild ohne sichtbare Übergänge zu generieren.

Sort-Last-Verfahren

Eine weitere Möglichkeit zur Realisierung von Parallelem Rendering ist das Sort-Last-Verfahren. Auch hier ist die Sortierung der Geometrien der Szene namengebend. Dies geschiet nach der Geomtriebearbeitung und Rasterisierung. Es wird der Ansatz verfolgt, nicht das finale Ausgabebild, sondern die 3D-Geometrien zu unterteilen. Demnach werden den Nodes Geometrie-Objekte zur Berechnung zugeteilt, unabhängig davon, wo sie sich in der Szene befinden. Fig.2 zeigt ein mögliches Cluster zur Realisierung des Sort-Last-Verfahrens.

Die Anwendung läuft auf jedem Application-Node. Diese senden die berechneten Bilddaten, die gerenderten Einzelobjekte der Szene, über das Netzwerk an den Display-Node. Hier wird die Ausgabe der einzelnen Daten synchronisiert und verknüpft. Um die Objekte zu einem Ausgabebild zusammenzufügen werden Pixel- und Tiefeninformationen benötigt. Der Austausch fertiger Bilddaten über das Cluster führt zu enormer Netzlast und kann sich negativ auf die Performance auswirken.

Chromium

Im Folgenden wird die Bibliothek Chromium, die an der University of Stanfort entwickelt wurde, näher betrachtet. Chromium unterstützt den Betrieb von OpenGL-basierten Graphik-Anwendungen unter Verwendung eines Render-Clusters. OpenGL ist eine Graphik API, die diverse Routinen zur Generierung und Darstellung komplexer 3D-Geometrien zu Verfügung stellt. Chromium erweitert die Funktionalität von OpenGL um zahlreiche Features zur Realisierung von Render-Clustern. Elementare Verfahren, wie Sort-First und Sort-Last, werden unterstützt. Es ist möglich die Konfigurationen ausschließlich in der Konfigurationsdatei vorzunehmen. Demnach kann jede OpenGL-basierte Anwendung ohne Modifikation über ein Render-Cluster betrieben werden.

Stream-Processing-Unit

Eines der zentralen Probleme im Zusammenhang mit Parallelem Rendering stellt die Verteilung der Graphikanweisungen über das Netz dar. Die einzelnen OpenGL-Kommandos müssen sinnvoll in Teilstreams zerlegt, und übers Netzwerk kommuniziert werden. Es ist zu gewährleisten, dass die Render-Nodes nur für ihre Darstellung relevante Anweisungen berechnen. Für das Sort-First Verfahren bedeutet dies, dass ein Render-Node nur die Geometrieanweisungen erhält, die zur Darstellung seiner Bildkachel rele-

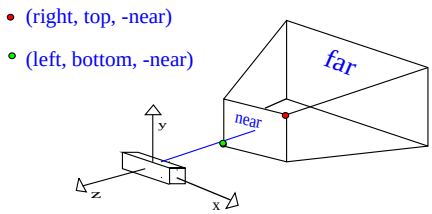


Abbildung 3: *Definition einer Sichtpyramide; roter und grüner Punkt werden relativ zum Kamerakoordinatensystem angegeben, near- und far-Ebene beschreiben die vordere und hintere Grenze der Sichtpyramide, sie werden entsprechend ihrer Entfernung zur Kamera angegeben*

vant sind. Um die netzwerkspezifischen Probleme beim Parallelen Rendering behanden zu können, stellt Chromium sogenannte SPUs (Stream Processing Units) zur Verfügung.

Eine SPU kann als Teilbibliothek von Chromium aufgefasst werden, die nicht automatisch zur Verfügung stehen. Durch bestimmte Anweisungen innerhalb der Konfigurationsdatei können SPUs auf Cluster-Nodes geladen, und ihre Funktionalität zur Verfügung gestellt werden. Verschiedenste Konzepte wurden realisiert, wie beispielsweise das Sort-First- und Sort-Last-Verfahren.

View- und Projektionsmatrix

Um das weitere Vorgehen nachvollziehen zu können, bedarf es einiger OpenGL-Grundlagen. In einer OpenGL-basierten Szene sind mehrere Koordinatensysteme von Bedeutung. Die Objekte der virtuellen Umgebung werden relativ zum Weltkoordinatensystem definiert. Die Kamera, aus deren Sicht die Szene dargestellt wird, befindet sich zu Beginn im Ursprung des Weltkoordinatensystems. Sie definiert ein weiteres Koordinatensystem, das Kamerakoordinatensystem. Ohne Modifikation dieses Systems ist die Ausrichtung der Achsen denen des Weltsystems identisch. Die positive x-Achse zeigt nach rechts, die positive y-Achse nach oben und der Blick ist entlang der negativen z-Achse gerichtet. Der Ursprung des Kamerakoordinatensystems definiert sich durch die Position der Kamera. Um den Blickwinkel ändern zu können, ist es möglich die Kamera, mit Hilfe der View-Matrix, relativ zum Weltkoordinatensystem zu transformieren. Alle in der Graphik benötigten Transformationen im dreidimensionalen Raum lassen sich durch die 4*4 View-Matrix angeben. Sie werden gesammelt aufmultipliziert, so dass alle Veränderungen der Kameraposition enthalten sind.

Die Projektions-Matrix ermöglicht die Bestimmung einer Projektion der Szene. Hier stehen Funktionen zur orthographischen und perspektifischen Projektion zur Verfügung. Mit Hilfe der perspektivischen kann der Szene mehr Räumlichkeit verliehen werden, weit entfernte Objekt werden kleiner dargestellt. Des Weiteren ist es möglich einen bestimmten Bereich der Szene im Fokus darzustellen. Anschaulich kann dies als eine Art Fenster aufgefasst werden, durch das man auf die Szene schaut. Die in Fig.3 dargestellten Parameter werden zur eindeutigen Definition eines Sichtfensters, auch Frustrum genannt, herangezogen. Sie sind relativ zum Kamerakoordinatensystem.

Konfigurationsdatei

Bei der Realisierung eines Render-Clusters unter Verwendung von Chromium spielt die Konfigurationsdatei eine entscheidende Rolle. Hier werden alle individuellen Variablen gesetzt, die für den Betrieb der Anwendung über das Netzwerk von Bedeutung sind. Es können Bildbereiche definiert, SPUs geladen und anwendungseigene Werte überschrieben werden. Die OpenGL-eigenen View- und Projektionsmatrizen können neu definiert, oder erweitert werden. Die Konfigurationsdatei stellt demnach die entscheidende Schnittstelle zwischen Anwendung und Netzwerk-Topologie dar.

Betrieb der Holobench unter Verwendung von Chromium

Einstieg

Im Folgenden wird der Versuch unternommen, das interaktive VR-System Holobench mit Hilfe von Chromium über ein Render-Cluster zu betreiben. Bei der Herangehensweise sind die Eigenschaften der Hardware zu berücksichtigen. Die Holobench stellt ein Tiled-Display-System dar. Charakteristisch sind in diesem Zusammenhang die zwei, im rechten Winkel angeordneten, Projektionsflächen. Demnach verteilt sich die Darstellung der Graphikanwendung auf zwei separate Fenster. Die Holobench wird mit Aktiv-Stereo betrieben und unterstützt Head-Tracking. Diese Funktionalitäten ermöglichen dem Betrachter einen immersiven 3D Eindruck. Das Objekt scheint vor den Projektionsflächen im Raum zu schweben.

Um den Funktionsumfang der Holobench gerecht zu werden, müssen die drei Hauptkriterien, Tiled-Display, Tracking und Aktiv-Stereo, für Paralleles Rendering umgesetzt werden.

Tiled-Display

Die zentrale Schnittstelle, um eine Applikation über Chromium zu betreiben, stellt die Konfigurationsdatei dar. Hier werden alle nötigen Einstellungen bzgl. der Parallelisierung vorgenommen. Da es sich bei der Holobench um ein mehrseitiges Tiled-Display-System handelt, wird das Sort-First-Verfahren angewandt. Eine spezielle SPU verteilt die OpenGL-Streams auf die Render-Nodes. Die Szene wird in zwei Bildbereiche unterteilt, die den Projektionsflächen der Holobench entsprechen. Diese Bereiche, auch Tiles genannt, werden jeweils von einem Node berechnet und separat über einen Projektor auf die Ebenen projiziert.

Die Koordinaten der Tiles werden relativ zu den Ausmaßen der Projektionsflächen definiert. In der Darstellung muss die Verzerrung, die durch den rechten Winkel der Projektionsflächen entsteht, berücksichtigt werden. Die Projektion auf die vertikale Fläche kann vorerst unverändert bleiben. Die untere Projektion muss die Verzerrung ausgleichen. Eine Rotation der View-Martix von 90° relativ zur x-Achse behebt dieses Problem.

Zu Beginn wird veranschaulicht, wie die Geometrie der zwei perspektivischen Sichtpyramiden angeordnet ist. Um später variable Kamerapositionen realisieren zu können, werden erst einmal Referenzwerte bestimmt. Der Betrachterpunkt entspricht der Position der Kamera, er ist für beide Frustren identisch. Der untere Öffungswinkel der oberen Sichtpyramide und der obere Öffnungswinkel der unteren Sichtpyramide betragen jeweils 45°. Somit kann gewährleistet werden, dass die Frustren exakt aneinander anschließen. Diese Bedingung muss in jedem Fall erfüllt sein, um die Darstellung der Szene ohne Lücke oder Überschneidung gewährleisten zu können. Die virtuelle Kamera befinde sich mittig vor der vertikalen Ebene. Die right, left, bottom und top-Werte zur Definition der Frustren werden relativ zu den Ausmaßen der Projektionsflächen festgelegt. Die genaue Anordnung der Referenz-Frustren ist in Fig. 4 dargestellt. Der far-Wert wird in Abhängigkeit der darzustellenden Objekte gesetzt.

Die Tiles werden jeweils durch Angabe des Ursprungs und der Pixelausdehnung in x- und y-Richtung definiert. Es handelt sich um konstante Werte, die von der Position der Kamera unabhängig sind. Auch hier ist der nahtlose Übergang der Tiles zu gewährleisten.

Tracking

Die Erfassung und Verfolgung bewegter Objekte im Raum wird Tracking genannt. Um die Immersion der Holobench zu steigern, wurde ein solches Tracking-System integriert. Ein Sensor, der an der Shutter-Brille des Nutzers installiert ist, ermöglicht zu jedem Zeitpunkt die exakte Erfassung der Betrachterposi-

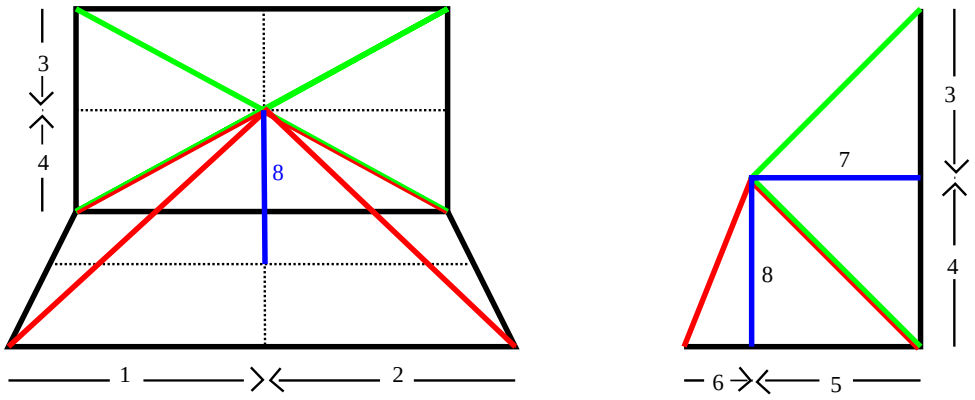


Abbildung 4: *Referenz-Frustren der Projektionsebenen, 1 left; 2 right; 3 top(oben); 4 bottom(oben); 5 top(unten); 6 bottom(unten); 7 near(oben); 8 near(unten); Die eigentliche Sichtpyramide befindet sich demnach hinter den Projektiosflächen. Die far-Ebene ist aus Gründen der Übersichtlichkeit nicht dargestellt*

tion. Die gemessenen Koordinaten werden permanent an das System übergeben und ermöglichen, durch anschließende Berechnungen, die Darstellung der Szene aus der individuellen Sicht des Nutzers. Konkret bedeutet dies, dass in Folge einer Kopfbewegung der Blickwinkel auf die Szene modifiziert wird. Dies geschieht im Idealfall so schnell, dass der Betrachter keine Verzögerung der aktualisierten Darstellung registriert. Die Position der virtuellen Kamera richtet sich stets nach den Trackerkoordinaten. Der Ursprung des Kamerakoordinatensystems wird fortlaufend aktualisiert, um im Auge des Betrachters zu liegen. Demnach wird die Bewegung des Betrachters durch Kameratranslationen simuliert. Anschaulich kann man sich vorstellen, dass die virtuelle Kamera fest am Kopf des Betrachters montiert ist.

Zur Umsetzung eines Tracking-Systems müssen, neben der nötigen Hardware, auch die permanenten Berechnungen der Kamerakoordinaten und Frustren unterstützt werden. Die Manipulation der Frustren sorgt dafür, dass die Projektionsflächen der Holobench im Fokus des Betrachters liegen. In Fig. 5 wird eine mögliche Anordnung der Sichtpyramiden dargestellt, die aus einer Bewegung nach links oben resultiert.

Unter Verwendung von Chromium gibt es zwei Möglichkeiten die View- und Projektions-Matrizen zu definieren. Zum einen wird die Veränderung der Matrizen innerhalb der Konfigurationsdatei unterstützt. Die hier vorgenommenen Transformationen und Projektionen werden auf die in der Anwendung definierten Matrizen multipliziert.

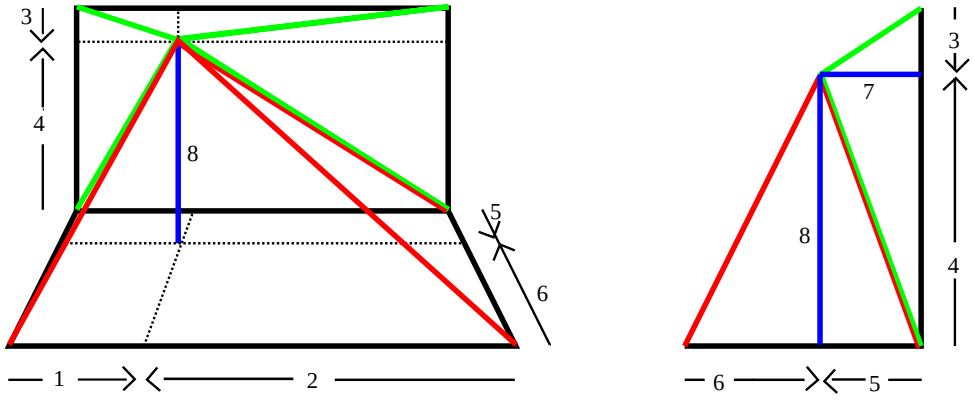


Abbildung 5: *Frustren für veränderte Betrachterposition, die Werte zur Definition der Sichtpyramiden müssen in Abhängigkeit zur Kamerabewegung neu bestimmt werden*

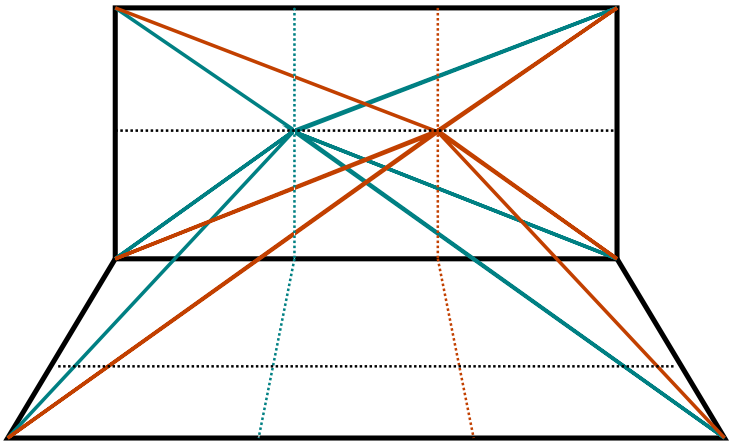


Abbildung 6: *Frustren für Stereo-Projektion, der Abstand der Sichtzentren ist zur Verdeutlichung größer dargestellt*

Innerhalb einer Konfigurationsdatei kann nicht mit dynamischen Daten gearbeitet werden, da die Datei zu Beginn des Aufrufs durchlaufen wird und feste Parameter für die gesamte Laufzeit setzt. Grundlegende Bedingung zur Aktualisierung der Blickwinkel ist das dynamische Verhalten der Kamerakoordinaten. Daher lässt sich innerhalb der Konfigurationsdatei keine Unterstützung des Tracking-Systems erreichen.

Zum anderen besteht die Möglichkeit die View- und Projektions-Matrizen innerhalb der Anwendung anzugeben. Hier ist die permanente Veränderung von Variablen möglich. Zu Beginn der Anwendung befindet sich die Kamera, entsprechend den Referenzwerten, zentriert vor der vertikalen Projektionsfläche. Für diese Position sind die Werte der Frustren fest bestimmt. Sie orientieren sich direkt an den Maßen der Projektionsflächen der Holobench. Mit Hilfe der Trackerkoordinaten kann nun die Position der Kamera permanent aktualisiert werden. Anzumerken ist, dass das gesetzte Frustrum, auf Grund der veränderten Kameraposition, neu berechnet werden muss. Der Fokus der Darstellung liegt stets auf den Projektionsflächen, so dass die neuen Werte mit Hilfe von Abweichungen relativ zu den Referenzwerten ermittelt werden können.

Zusammenfassend lässt sich festhalten, dass für jedes Tile die View- und Projektions-Matrix fortlaufend neu berechnet werden. Die Unterstützung des Tracking-Systems ist nur durch Modifikation der Anwendung zu erreichen. Die Anweisungen innerhalb der Konfigurationsdatei reichen in diesem Fall nicht aus.

Stereo-Projektion

Die Immersion der Holobench wird hauptsächlich durch den Einsatz von Stereo-Projektion erreicht. Die menschlichen Augen stehen in einem geringen Abstand zueinander und ermöglichen die zeitgleiche Betrachtung unserer Umwelt aus schwach voneinander abweichenden Perspektiven. Der Abstand der Augen befähigt den Menschen dreidimensional zu sehen. Um diesen Effekt in der VR simulieren zu können, wird die Szene ebenfalls aus zwei, leicht voneinander abweichenden, Perspektiven gerendert. Es gibt zwei grundlegende Verfahren, mit denen sich Stereo realisieren lässt, Aktiv- und Passiv-Stereo. Bei der Holobench wird Aktiv-Stereo angewandt. Die Projektoren stellen in minimalen Zeitintervallen jeweils das rechte und linke Bild dar. Der Betrachter trägt eine Shutter-Brille, die in identischer Frequenz jeweils das linke, bzw. rechte Auge verdeckt. Auf diese Weise nehmen die Augen nur die aus ihrer Sicht richtige Perspektive wahr. Die dreidimensionale Wahrnehmung ist simuliert.

Aktiv-Stereo muss von Hard- und Software unterstützt werden. Analog zum Tracking bietet Chromium die Möglichkeit diese Anweisungen innerhalb der Konfigurationsdatei oder der Anwendung vorzunehmen. Da die Stereo-Darstellung in Abhängigkeit zur Betrachterposition steht, ist in Verbindung mit

Tracking nur die Realisierung innerhalb der Anwendung möglich.

Obwohl Chromium Aktiv-Stereo in Verbindung mit Tracking noch nicht unterstützt, wurde der Ansatz theoretisch betrachtet. Zu einem späteren Zeitpunkt müssen lediglich kleine Änderungen vorgenommen werden, um dem vollständigen Funktionsumfang der Holobench gerecht zu werden.

Der Betrieb der VR-Rotunde unter Verwendung von Chromium

Im Folgenden soll der Betrieb der VR-Rotunde mit Hilfe von Chromium näher beleuchtet werden. Das VR-System besteht aus einer zylinderförmigen Leinwand, die eine Panoramadarstellung von Graphikanwendungen ermöglicht. Allgemein bringt die Umsetzung keine großen Probleme mit sich. Die Ausgabe geschieht nach dem Tiled-Display-Prinzip, was den Einsatz des Sort-First-Verfahrens nahelegt. Die Justierung der Projektoren spielt eine zentrale Rolle, da die Übergänge der Bildbereiche unsichtbar zu realisieren sind. Für die Stereo-Darstellung wird Passiv-Stereo herangezogen, was die doppelte Anzahl an nötigen Projektoren mit sich bringt. Da die VR-Rotunde kein Tracking unterstützt, können die View- und Projektions-Matrizen innerhalb der Konfigurationsdatei fest definiert werden. Eine Modifikation der Anwendung ist unter diesen Voraussetzungen nicht erforderlich.

Fazit

Zusammenfassend kann festgehalten werden, dass der Aufwand, Paralleles Rendering zu betreiben, stark an das jeweilige VR-System gebunden ist. Mit Hilfe von Bibliotheken wie Chromium ist die generelle Parallelisierung von Graphikanwendungen schnell umzusetzen. Meist ist dies ohne Modifikation der Anwendung möglich. Handelt es sich aber um spezielle Systeme, wie beispielsweise eine Holobench, müssen einige Details bedacht und umgesetzt werden. Kritisch muss betrachtet werden, dass Chromium die Kombination von Aktiv-Stereo und Tracking bislang nicht unterstützt.

Danksagung

Im Folgenden möchte ich einigen Personen danken, die mir die erfolgreiche Teilnahmen am Gaststudentenprogramm ermöglicht haben. Besonderer Dank gilt meinem Betreuer Herrn Dr. Herwig Zilken, der mir bei offenen Fragen stets mit Rat und Tat zur Seite stand. Herrn Dr. Rüdiger Esser möchte ich für die erfolgreiche und vor allem persönliche Organisation danken. Das Empfehlungsschreiben, das mir Herr Prof. Dr. Stefan Müller der Universität Koblenz ausstellte, war der erste, grundlegende Schritt zur Teilnahme an diesem Programm. Hierfür möchte ich auch ihm herzlich danken.

Literatur

1. Chromium; http://chromium.sourceforge.net
2. OpenGL; OpenGL Programming Guide: Mason Woo, Jackie Neider, Tom Davis
3. ViSTA's World, Unstructured Manual, Herwig Zilken, ZAM Forschungszentrum Jülich GmbH, 1999

Erweiterung der Cluster-Middleware ParaStation zum Ressourcenmanager in UNICORE

Tobias Hertkorn

Universität Bayreuth
Fachbereich Physik

E-Mail: tobias@hertkorn.com

Zusammenfassung:
Der Gedanke hinter Gridsystemen ist, dass Computing Grids die Bereitstellung von Rechenkapazität so einfach machen soll, wie das “power grid”, also das Stromnetz, die Bereitstellung von Strom gemacht hat. Angepasste und für ein Grid geschriebene Programme sollen an jeder “Steckdose” im Grid funktionieren. Ziel der Arbeit am Projekt “ParaMeta”[1] war die Cluster-Middleware ParaStation möglichst einfach über die Grid-Software UNICORE ansprech- und nutzbar zu machen.

Vergleich zwischen Cluster und Grid

Um die Begriffe Cluster und Grid sauber trennen zu können, muss man diese genauer definieren. Dazu wird in diesem Abschnitt auf die Gemeinsamkeiten und die Unterschiede zwischen den beiden Architekturen eingegangen. Im Speziellen werden hier die Varianten besprochen, die zum Verarbeiten von rechenintensiven Arbeiten verwendet werden: High-Performance Cluster (HPC) und Rechengrids (Compute Grids).

Gemeinsamkeiten

“Ein Computercluster, meist einfach Cluster, von engl. cluster = Traube, Bündel, Schwarm, genannt, bezeichnet eine Anzahl von vernetzten Computern, die zur parallelen Abarbeitung von zu einer Aufgabe gehörigen Teilaufgaben zur Verfügung stehen. [...] die Lastverteilung [findet] auf der Ebene einzelner Prozesse statt (mindestens ein Prozess wird auf einen Rechner verteilt), die auf einer oder verschiedenen Maschinen des Clusters gestartet werden.” [2]

“Grid-Computing (englisch grid computing = “Gitterrechnen”) bezeichnet alle Methoden, die Rechenleistung vieler Computer innerhalb eines Netzwerks so zusammenzufassen, dass über den reinen Datenaustausch hinaus die (parallele) Lösung von rechenintensiven Problemen ermöglicht wird (verteiltes Rechnen). Jeder Computer in dem “Gitter” ist eine den anderen Computern gleichgestellte Einheit.” [3]

Man kann sehen, dass Grid und Cluster viele Gemeinsamkeiten haben. Beide Architekturen werden benötigt um große Arbeits- oder Datenmengen zu verarbeiten. Dies wird bei den rechenintensiven Anwendungen durch das Aufteilen der Arbeit in einzelne Prozesse und das Verteilen dieser Prozesse auf unabhängige Computer erreicht. Doch wo liegt der Unterschied zwischen den beiden Architekturen?

Unterschiede

Der Hauptunterschied liegt darin, dass Gridsysteme administrative und architektonische Grenzen überschreiten. Das heißt Grids schließen heterogene Ressourcen zusammen. Damit sind vor allem verschiedene Plattformen und Hardware-/Software-Architekturen gemeint, die an unterschiedlichen Orten und unter verschiedenen Administratoren laufen. Im Gegensatz zu Cluster, die by Design eine Einheit bilden können Grids erst durch Virtualisierung der Ressourcen zu einer logischen Einheit verschmolzen werden. Dies deckt die Überschreitung von architektonischen Grenzen ab.

Die Überschreitung von administrativen Grenzen erfordert eine große Unabhängigkeit der im Grid integrierten Sites. Das Grid darf nicht auf das Vorhandensein einer speziellen Site angewiesen sein. Beziehungsweise administrative Einheiten müssen jederzeit in der Entscheidung frei sein ihre Ressourcen dem Grid zur Verfügung zu stellen oder zu sperren.

UNICORE

Diese angestrebte Virtualisierung und Autonomität wird von vielen konkurrierenden Systemen erreicht. UNICORE ist eines dieser Systeme. Es ist ein standardisiertes Open-Source Grid System, das Supercomputer und Cluster zu einem Grid vereinigt. Zusätzlich sorgt es für die Sicherheit und Eigenständigkeit der im Grid beteiligten Sites.

Ziele von UNICORE

Das Hauptziel von UNICORE ist es einen nahtlosen, sicheren und intuitiven Zugang zu verteilten Ressourcen zu schaffen. UNICOREs Architektur erlaubt es durch Virtualisierung die unterschiedlichen Hardwaresysteme, Betriebsysteme, Batchsysteme, sich unterscheidende Anwendungsumgebungen, Namenskonventionen und Storagestrukturen zu verstecken. Des Weiteren wird durch Authentifizierung unter Einsatz von Zertifikaten eine einheitliche Login- und Securitymethode zur Verfügung gestellt.

Dadurch wird es Forschern und Ingenieuren möglich die von UNICORE bereitgestellten Ressourcen zu verwenden, ohne die hersteller- oder bereitstellerspezifischen Details zu kennen. Dies wird durch eine grafische Oberfläche zum Erstellen, Abschicken und Überwachen von komplexen Jobs weiter erleichtert. Diese GUI unterstützt zusätzlich beim Übertragen der Daten und des Programms zum Zielsystem im Grid. Idealerweise kann ein einmal mit dieser Oberfläche erstellter Job ohne weitere Anpassung auf einem beliebigen Zielsystem im UNICORE Grid laufen.

Architektur

Um die geforderte Virtualisierung zu realisieren und die Autonomität der ins Grid integrierten Sites zu gewährleisten wurde bei UNICORE eine Architektur entwickelt, die aus mehreren Schichten besteht.

Für den Benutzer sind dabei 3 Schichten sichtbar und relevant:

- UNICORE Client
- UNICORE Site (Usite)
- Virtual Site (Vsite)

Wie in Abb. 1 zu sehen setzt sich eine Usite (blaue Umrandung) aus einem Gateway und einer oder mehreren Vsite (gelbe Umrandung) zusammen. Das Gateway ist für die Überwachung der Logins zu

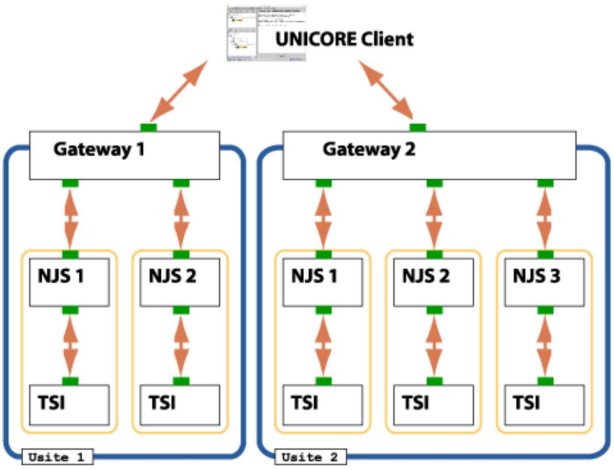


Abbildung 1: Schema der Architektur von UNICORE

den Vsites zuständig. Die Vsite ist für den Benutzer die virtuelle Repräsentation des darunterliegenden Cluster oder Supercomputer. Für den Benutzer ist die Vsite die kleinste sichtbare Einheit.

Innerhalb der Vsite-Schicht geschieht das Konvertieren der virtualisierten Jobansicht in die konkrete Umsetzung für die jeweilige Architektur. Um diese zu bewerkstelligen besteht eine Vsite wiederum aus zwei Schichten. Im generischen Network Job Supervisor (NJS) wird unter anderem die Konvertierung vorbereitet und an das extra auf die Zielarchitektur/-software zugeschnittene Target System Interface (TSI) weitergereicht. Dieses TSI wird in dem von uns eingesetzten Szenario ParaStation ansprechen und steuern.

ParaStation

Einführung

ParaStation [4] ist eine von der ParTec Cluster Competence Center GmbH betreute und weiterentwickelte Cluster Middleware Lösung. Sie setzt auf MPIch auf, einer Open-Source Implementation des Message-Passing Interface Standards.

Zusätzlich bietet ParaStation ein integriertes Prozessmanagementsystem. Dieses kann auch als rudimentäres Batchsystem eingesetzt werden. ParaStation kümmert sich um die Prozessverteilung und -überwachung im Cluster und kann verwaiste Prozesse automatisch erkennen und terminieren. Daneben hat es auch einen Mechanismus Einprozessorjobs auf dem Cluster zu verteilen.

Grundinstallation ParaStation mit UNICORE

Installation

Bei der Ideenfindung für ParaMeta wurde auf eine bereits bestehende ParaStation Installation zurückgegriffen. Es gab ein designiertes Frontend in diesem Cluster, auf dem die UNICORE Server Installation durchgeführt wurde.

Als Startpunkt wurde das Quickstart Bundle[5] mit dem NO_BATCH System als TSI installiert. Eine Installation des Quickstart Bundle auf dem Cluster-Frontend ist einfach und bietet bereits die Möglichkeit MPI-Jobs auf dem von ParaStation gemanagten Cluster via UNICORE zu verteilen.

Probleme

Um einen Job über die beim UNICORE Client standardmäßig mitgelieferten Plugins zu kompilieren und zu starten muss man die genauen Pfade zu den Compilern kennen. Außerdem kann man als Benutzer von ParaStation über Umgebungsvariablen steuernd in den Verteilungsmechanismus von dem integrierten Prozessverteilungssystem eingreifen. Diese müssen dem Benutzer bekannt sein, um das volle Potential von ParaStation nutzen zu können.

Dadurch fehlt die von UNICORE angestrebte Abstraktion von Jobs fast vollständig und bietet außer den einheitlichen Authentifizierungsmechanismen keinen echten Mehrgewinn gegenüber SSH.

ParaMeta

Der UNICORE Client bietet eine API[6] um Plugins für diesen zu schreiben. Um ParaStation besser in UNICORE zu integrieren und damit die Abstraktion der MPI-Jobs wieder zu erhöhen wurde das Client-Plugin ParaMeta entwickelt.

Motivation

Der Grundgedanke von UNICORE verlangt nach Virtualisierung der Ressourcen. Deswegen ist das Ziel dieses Projekts im ersten Schritt ParaStation für den Enduser einfacher ansprechbar zu machen. Des Weiteren soll es komfortabel möglich sein die Hauptparameter eines MPI-Jobs zu beeinflussen: Die Anzahl der angeforderten CPUs und die MPI-Implementation, die benutzt werden soll, um den Job zu starten.

Zusätzlich sollen steuernden Umgebungsvariablen über das Plugin setzbar sein. Vor allem der Parameter PSI_OVERBOOK ist dort besonders hervorzuheben. Dieser beeinflusst, ob mehrere Prozesse auf einer CPU laufen dürfen. Damit beeinflusst dieser Parameter auch direkt das Plugin. Denn damit ist das Plugin nicht mehr an die Ressourcendefinitionen der Vsite gebunden. Wenn PSI_OVERBOOK gesetzt ist muss es auch möglich sein eine beliebige Anzahl von CPUs anzufordern und damit die von UNICORE gegebene maximale Anzahl von CPUs zu ignorieren.

Anpassungen UNICORE Client

Der UNICORE Client ist in Java geschrieben. Dadurch wurde auch das Plugin in Java entwickelt und über die Plugin-API vollständig in die Erstellung eines Jobs integriert (Fig. 2). Dadurch stehen für das Plugin alle gewohnten Werkzeuge zur Verfügung. ParaMeta kann als vollwertiger Schritt in einer Flow-Definition eines Jobs oder Unterjobs eingesetzt werden. Es unterstützt ebenfalls alle Eigenschaften, um bedingte Schritte einzuleiten. Das heißt es können über das If-Then-Else Plugin Entscheidungsbäume basierend auf den Returnwert des MPI-Programms erstellt werden.

Der Hauptscreen (Fig. 3) des Plugins zeigt die einfach und intuitiv zu bedienenden Grundsteuerelemente des Plugins. Über den Schieberegler oder dem Textfeld kann die benötigte Anzahl von CPUs eingestellt werden. Über die Dropdown-Liste kann die MPI Ressource ausgewählt werden, die für das Abschicken des Jobs verwendet werden soll. Bei “Command Line” wird der Name des auszuführenden Programms eingetragen. Zusätzlich können über die Zeile “Environment” beliebige Umgebungsvariablen definiert werden, die an den MPI-Job weitergegeben werden.

ParaMeta ist es auch möglich eine rudimentäre Überprüfung des Jobs durchzuführen. ParaMeta macht dafür eine Validierung der Einstellungen und kann somit dem Benutzer schon vor dem Abschicken des Jobs auf mögliche Probleme aufmerksam machen.

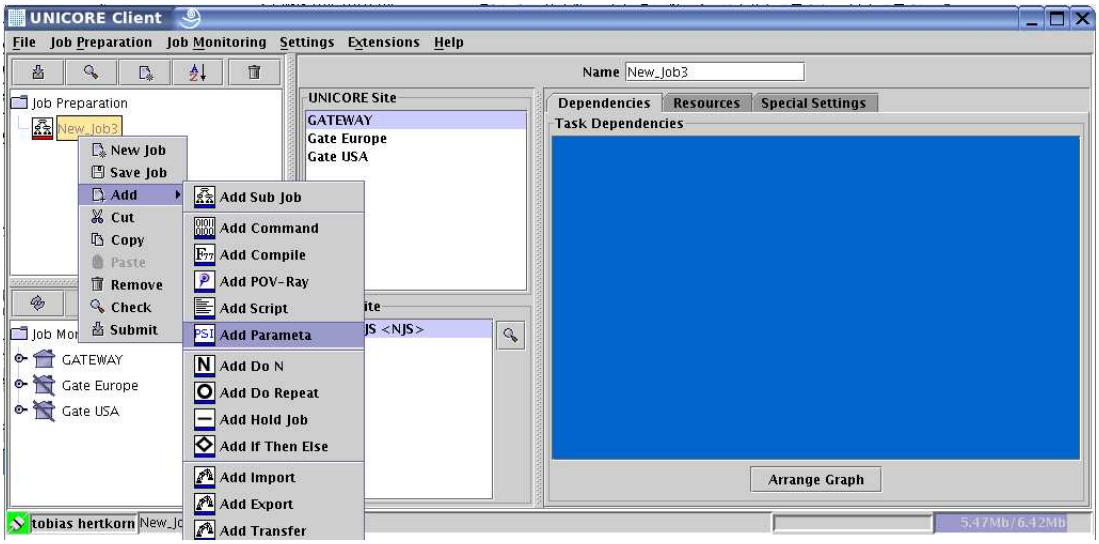


Abbildung 2: Hinzufügen des ParaMeta-Moduls in einen Job

Um die Anforderung zu erfüllen, dass verschiedene MPI-Versionen auswählbar sein sollen, wurden sogenannte Software-Ressourcen in der UNICORE serverseitigen Architektur definiert.

Anpassungen UNICORE serverseitig

Um die Pfade zu den MPI Implementationen und die exakten Jobsubmit Kommandos vom Benutzer zu verstecken müssen auch an der serverseitigen UNICORE Installation Anpassungen vorgenommen werden.

Die Lösung, die für ParaMeta gewählt wurde ist die Definition von Softwareressourcen in der Incarnation Database (IDB)[7]. Jede definierte Softwareressource repräsentiert eine spezielle MPI Lösung, die dem Benutzer zur Auswahl bereitgestellt werden soll.

Die spezielle Softwareressource für ParaStation ist sehr einfach gehalten:

```
#!/bin/bash
PATH=$PWD:$PATH
if [ $PSI_NROFNODES -gt 1 ]; then
exec $* -np $PSI_NROFNODES
else
exec /opt/parastation/bin/psmstart $*
fi
```

Die Umgebungsvariable PSI_NROFNODES hält die Anzahl der angeforderten CPUs. Falls diese gleich eins ist, wird angenommen, dass kein MPI Job abgeschickt werden soll, sondern ein normaler Job auf einem der freien CPUs im Cluster gestartet werden soll.

Durch den Mechanismus der Software-Ressource sind jegliche Pfadabhängigkeiten und damit systemspezifische Anpassungen einmal zentral vom jeweiligen Administrator zu konfigurieren. Sobald das geschehen ist, kann ein Job, der für ParaMeta vorbereitet wurde, ohne Änderungen auf jeder Vsite gestartet werden, die an ParaMeta angepasst ist. Dadurch ist ein erster wichtiger Schritt in Richtung Virtualisierung von MPI-Ressourcen gemacht worden.

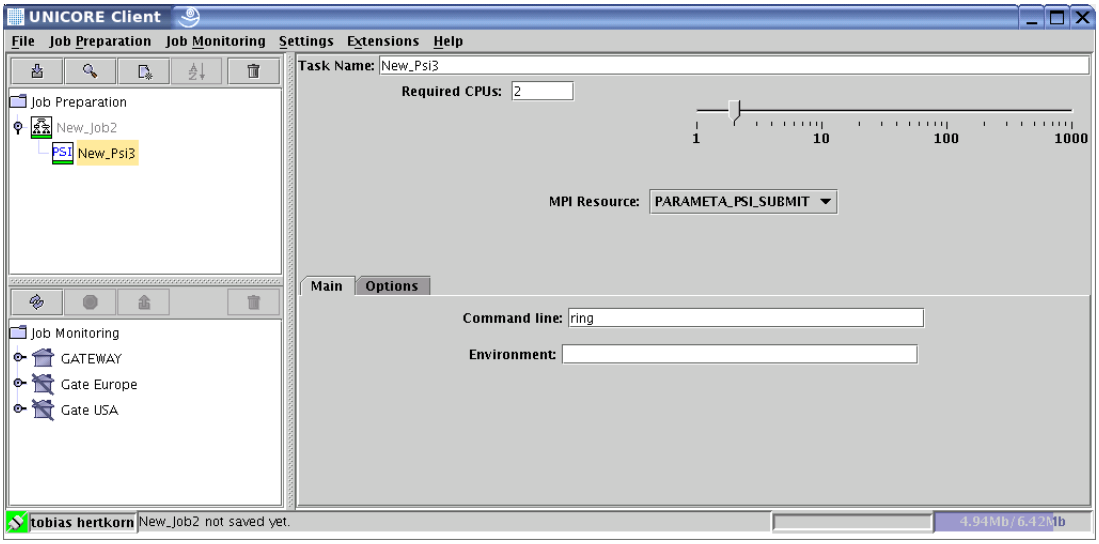


Abbildung 3: Hauptscreen von ParaMeta

Ausblick

Als zweite Phase der Integration von ParaStation in das UNICORE Umfeld ist geplant ein dem ParaMeta Plugin entsprechendes automatisches Make-Plugin zu schreiben. Dieses soll eingesetzt werden, um MPI Programme vollautomatisch mit einer MPI-Ressource zu kompilieren. Diese kompilierten Programme werden anschließend mit ParaMeta gestartet. Danach werden in einer dritten Phase beide Plugins so umgeschrieben, dass sie mit einer breiten Palette von MPI Implementationen arbeiten.

Literatur

1. ParaMeta - A UNICORE Plugin for ParaStation
   http://www.hertkorn.com/parameta/
2. Wikipedia - Computercluster
   http://de.wikipedia.org/wiki/Computercluster
3. Wikipedia - Grid-Computing
   http://de.wikipedia.org/wiki/Grid
4. ParaStation
   http://www.parastation.com/
5. UNICORE at sourceforge - Quickstart bundle!
   http://unicore.sourceforge.net/unicoreatsf.html
6. UNICORE Client JavaDoc
   http://unicore.sourceforge.net/docs/javadocs/Client/index.html
7. IDB Manual
   http://unicore.sourceforge.net/docs/idb_manual.pdf

Simplified Certificate Management in Grid PKIs

Martin Hoch

University of Applied Sciences Würzburg

E-mail: martin@littlerobot.de

Abstract: This paper deals with a simplified certificate management approach for the purpose of authenticating Grid participants. After introducing some basics of security, authentication and certificate management, we analyse the authentication and registration system used in a present-day Grid system such as UNICORE. We present a simplified certificate authentication and registration system (SCAR). This solution is simpler, easier to use, more manageable, offers fewer possibilities for human mistakes and therefore improves the system as a whole.

Introduction

In modern computer science security is a very important topic. In Grid computing, a single compromised user account can cause a considerable amount of damage. Consumed computer time, theft of scientific results or plain vandalism are just the obvious risks.

Therefore security is one of the most important non-functional qualities of a computer system and can be equally ranked with usability. Neither a system with bad usability nor with bad security can be properly used. These two aspects are in fact tightly connected. Higher usability increases security because the users won't make critical mistakes. On the other hand, improved security should increase the usability of a system by taking unnecessary decisions away from the user. Consequently, the security of a system can be improved by increasing its usability.

While designing systems one must also keep in mind that security is a tradeoff between a specific amount of security you get and the amount of money, flexibility, usability or the like that must be given up. A highly secured computer with no network access might be very secure, but is rather unusable.

Due to the fact that security is not a state but a process, it is necessary to permanently analyse, evaluate and evolve the security of a system.

We analysed the authentication system used in the UNICORE Grid middleware in terms of security and usability. In doing so we found a possibility to increase the functionality, usability and security of our current authentication system.

This approach makes use of the fact that we can control both server and client code.

The remainder of the paper is organised as follows: First we will introduce the current UNICORE authentication system, then discuss the downsides of the current authentication system, and finally introduce a simplified approach for certificate management.

Authentication mechanism in current Grid systems

At the moment UNICORE, like various other distributed systems, uses a certificate based system to authenticate all Grid participants. Participants are users and server components (NJS and Gateway). In

doing so a public key infrastructure is created.

Public key infrastructure

In a public key infrastructure all certificates are ordered in a hierarchical tree structure. This means if you decide to trust the root of this structure and all certificates are signed properly, you can determine whether you can trust a particular other certificate. For that reason it is very important to deliver the trust anchor in a manner that either allows the user to verify its integrity or that makes it technically impossible that the user receives a wrong root certificate. From a usability point of view the second is preferred. Currently most public key infrastructures use one of the following two methods. The first method is to place the certificate on an SSL protected web-page where the server certificate is signed by a public CA like Verisign or Thawte. The other method is to use a certificate which is part of the public key infrastructure (PKI) itself. Both methods are highly vulnerable against man- in-the-middle (MITM) attacks. While the second method does not provide any security at all, the first is at least as secure as HTTPs. But how secure is this ?

HTTPs security

Although this question is beyond the scope of this paper, it is crucial for the security of the whole system. HTTPs has one major problem. Average users are trained to accept any certificate and ignore every warning which occurs. Even though the server could use a certificate signed by a well known certificate authority, the user has no possibility to distinguish between a standard self signed certificate and a forged certificate from a MITM attack. Therefore the user is responsible for the security of the system. But can we really make him responsible?

This problem is recognised and there is work in progress[4] to find a solution but it might never be solved completely.

For this reason the use of HTTPs should be avoided in the system if possible. In our case it is possible because we have control over client code and can therefore implement a system which does not rely on third party software.

Getting a user certificate

As mentioned in the introduction, the crucial point in modern computer security is often the human in the loop[1]. Therefore we look at the users part in the authentication system currently used in UNICORE.

For obtaining a certificate the user must follow these steps:

- Generate certificate request.
- Save the request as file.
- Upload the request to the CA.
- Fill out, sign and fax a document to the CA, which then emails the certificate to the user
- Save the certificate received by email as file.
- Import the certificate in the client.

Without going into detail one can see the main problem: most users do not understand the meaning of all these steps and therefore can make critical mistakes or can be induced to make critical mistakes by an attacker.

Summing up, we can identify two critical points in current certificate management: the *use of HTTPs for certificate distribution* and *too much user influence*. To solve these problems, a system is needed which combines usability and security in an optimal fashion.

To do so, we will first identify how a user friendly authentication system must look like.

User friendly authentication

Users are used to authenticate with usernames and passwords. Internet service providers are sending username and password pairs by mail. Banks are doing almost the same. They include additional one time passwords, the well known TANs. In a lot of other examples like E-Mail and most online-services the user authenticates with username and password.

By considering this, it is easy to assume that users are at least used to authenticate with usernames and passwords. Although passwords are called dead every year, and not without reason, the ease of use and distribution of passwords is still unbeatable. That's why we ignore all security flaws of passwords like phishing for a moment and focus on the easy initialisation and usability of such a system.

SCAR - Simplified Certificate Authentication and Registration

In this section we introduce a simple certificate authentication and registration system (SCAR).

The usability benefits of a password based authentication system is obvious. However, there are considerable security flaws. On the other side, the current certificate based authentication system is quite secure but has some problems with the registration and initialisation of new users, together with the secure delivery of the initial root CA certificate.

The SCAR system

SCAR combines the usability of authentication using username and password with the security of public key infrastructures.

To achieve this, SCAR implements a additional layer which is responsible for the registration and initialisation of new user accounts. This layer uses a three step process, which is shown in Fig. 1.

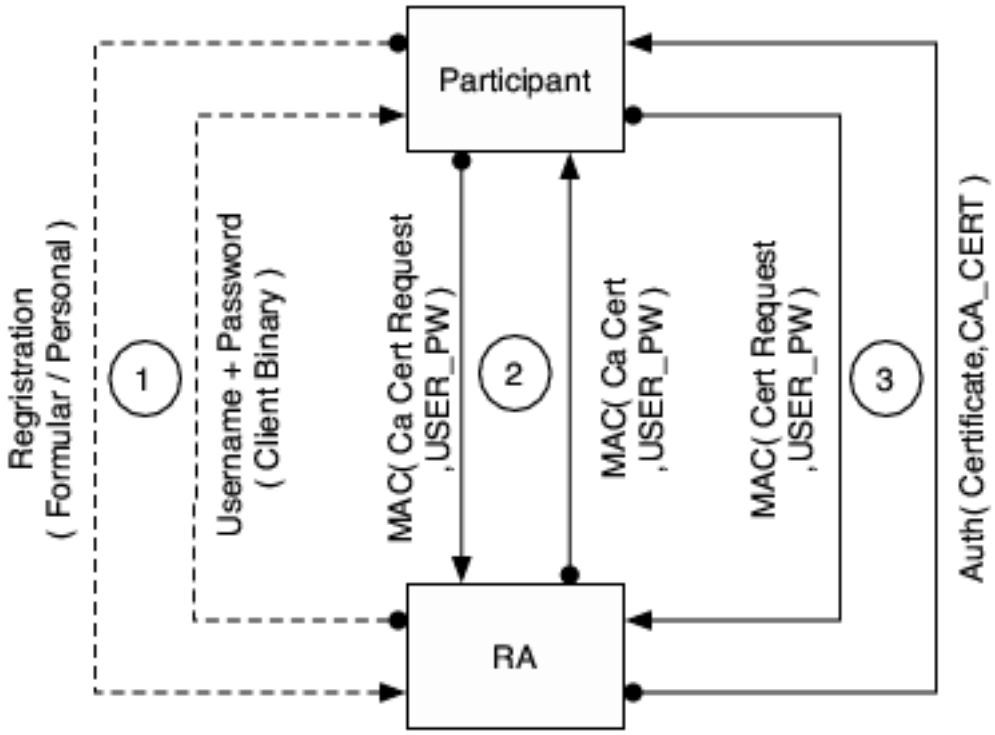


Figure 1: Simplified Certificate Authentication and Registration

Step 1

First of all the user must register with an authority which is responsible for the set of resources the user wishes to use. It is crucial to understand that this is always the first step for the user. The cases where users obtain the client before registering for Grid access are rather minor and even then, the client has no possibility at all to advice the users how to register. The only possibility would be, if either this information is hard coded or some information to identify the supplier of this information (e.g, certificates). All other methods would be vulnerable against MITM because the source of this information can not be trusted. The reason why hard-coding is a bad solution will be discussed in section .

In the first step of SCAR three tasks are performed: Firstly, the user must prove his identity to the registration authority. Secondly, the user must sign all necessary agreements. And thirdly, the registration authority must send the initial user information (e.g., username and password) through a secure out-of-band communication channel.

Therefore the following to way communication is used: The user registers for access and the registration authority responses with initial account information.

The registration authority is responsible that only the person which claims to have issued the registration is able to receive the response. This ensures password security and can be done by building on top of an existing relation with the user. This relation must exists or must be established anyway because issuing certificates to strangers makes no sense. During the registration the user also signs all necessary agreements.

The definite identification depends heavily on the kind of the relation and on the amount of security wanted. In general, the more personal this is done, the more certain the identity is. The best solution is to identify every user personally through his id-card. Using interoffice mail would also be very secure and

identifying your cubicle neighbour is of course not necessary. But it is not always that simple. Normally this identification is done through mail. This is, also not really secure, good enough for most real-life systems, so it should also be enough for our system.

The only thing which must be verified by the registration authority is whether the requester really is who he claims to be, and if the postal address is correct.

The initial account information from the registration authority contain at least username and password, but can also include system information such as the Grid gateway address, the RA address, the required software or others. Therefore a user account is created by the RA before it sends out the response.

With RA we always mean the computer contrary to the organisation we call registration authority.

After obtaining the response from the registration authority, the user enters that information into the client. With this information the client is able to automate the following two steps. This means that from this moment no more user actions are required.

Step 2

As prerequisites, the data from step one have to be available. There is no need for a trusted relationship with any Grid participant, but the client needs to know the address of the registration authority. This can be preconfigured or entered manually by the users.

The client now sends an initial request to the RA and receives a list of trusted certificates. These two messages are signed with a MAC[5] (message authentication code), calculated from the users password. Therefore spoofing is impossible since only the RA which issued the username and password is able to calculate the proper signature for the response. That's also the reason why it doesn't matter where the client gets the address information for the RA from. The communication should be done with CMP (Certificate Management Protocols) which is defined in RFC2510[7].

Step 3

From Step 2 the client got the CA certificate, and is therefore able to authenticate other Grid participants. Now the client needs to request its own certificate. To do so, he sends a Certificate Request to the RA. This request is again signed with a MAC calculated from the users password. In reply the RA sends the users certificate signed with its own key, and encrypted with the client's public key. The encryption is used as a proof of possession of the associated private key.

After this three steps the client is successfully registered and has obtained the necessary certificates for safe usage of the Gird PKI. Thereby the user had only to enter his username and password, which he has received through out-of-band communication channels. The rest is handled automatically.

SCAR: analysis and evaluation

In this section we note a few security issues resulting from the SCAR scheme outlined above.

The registration authority has full control over the certificates. This offers many possibilities and advantages, most notably, the "distinguished name" entry of the certificate can be formatted consistently. Most of the responsibilities for generating correct certificates rest on the registration authority, which is how it should be.

The user on the other side has only few responsibilities. He must file the application and enter access data into the client. This is simple and done in a way the user knows and understands. The possibilities for errors are vastly reduced. Therefore the user is no longer a major point of failure.

Although the system is already more secure than the current authentication method, there are still several remaining vulnerabilities.

A major problem with password authentication is that anyone who got the password is able to authenticate as the user, and is therefore able to initialise the system. Due to the fact that the password is used only for initialisation purposes, we can reduce the risk through two restrictions on the use of the password:

- make the password usable only once
- set the certificate validity to start in the future (i.e. with some days offset)

The first point allows the system to detect if someone else authenticates with the users password. Therefore the system is potentially affected for the time between the attacker's initialisation and the user's attempt to initialise.

This time period cannot be limited because the user can't be forced to use his access. Nevertheless we can try to reduced this period of time by asking the user to initialise as soon as possible and by delaying the use of the new certificate by a certain amount of time through setting the validity. To force the user to initialise within a certain period of time might be dangerous because it can go wrong in the case if an attacker as initialised successfully and the user would not try to do so because the deadline has already passed.

Advantages over hard-coded root certificates

Another common solution for the registration and authentication problem is to hard-code the CA's root certificate into the client application. This solves the initial thrust problem but is not applicable for a system like UNICORE [2]. The first argument against this solution is based on the fact, that Unicore is an open source project which is used for different Grids like OpenMolGrid[3] or Daisa[6]. Therefore the client is used in different environments with different certificate authorities responsible. Obviously it is not possible to hard-code all those certificates. Nevertheless it's possible to solve this problem by enabling the exchange of this trusted certificates during the deployment. Thus this is not yet major a argument.

But the fact that one client would not be able to operate in different certificate authorities scopes makes this problem more serious. In addition such a system in general is static and inflexible.

However the point at issue is that even with hard-coded certificates, the user still needs to register and the registration authority must authenticate him. Thereby all problems mentioned in Step 1 must be solved and the proof of possession satisfied. For this the software would at least needs a complexity equal to SCAR. Because the user must authenticate against the RA and the registration must be validated and compared to the request.

Therefore this has a higher complexity but no advantages compared to SCAR.

Summary and Outlook

Based on an analysis of current Grid authentication schemes and certificate management solutions, we have proposed a simple yet secure scheme for registering new users, distributing CA certificates and requesting user certificates.

The proposed SCAR scheme is simple and transparent for the user, vastly reducing the possibilities for errors. It's also fully compatible with the current UNICORE authentication system. Only the client must be altered to use SCAR and the RA must implement the necessary functionality. No server component, neither Gateway nor NJS must be adjusted.

On the other hand, the SCAR scheme is more secure than the present solutions, therefore the better usability does not imply a reduced level of security.

This concept can be pushed even further. It might be possible to eliminate the problem of certificate revocation through a system similar to Kerberos which uses automated certificate renewal to issue certificates with a reduced validity period. But this needs further research.

A Java implementation for easy integration of this scheme into existing applications such as the present UNICORE client is under way.

References

1. Humans in the Loop S.W. Smith IEEE Security and Privacy May 2003 http://www.cs.dartmouth.edu/ sws/papers/humans.pdf
2. UNICORE homepage: http://unicore.sourceforge.net/
3. OpenMolGRID homepage: http://www.openmolgrid.org
4. Hardening Web Browsers Against Man-in-the-Middle and Eavsdropping Attacks http://www2005.org/cdrom/docs/p489.pdf
5. Message Authentication Codes http://www.rsasecurity.com/rsalabs/node.asp?id=2177
6. DEISA homepage: http://www.deisa.org
7. Certificate Management Protocols: http://www.ietf.org/rfc/rfc2510.txt

Modern Methods in Protein Simulations

Christoph Junghans

Institute of Theoretical Physics
Department of Physics and Geoscience
University Leipzig

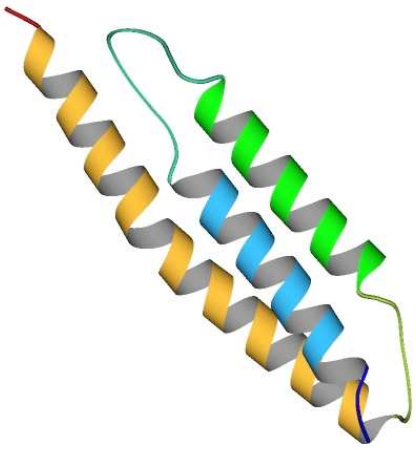
E-mail: christoph.junghans@itp.uni-leipzig.de

Abstract: I studied a realistic protein model using four advanced Monte Carlo techniques: multicanonical Monte Carlo, parallel tempering, Wang-Landau sampling and simulated tempering. The comparison showed that there are problems with ergodicity in the small energy region of big peptides due to the size of the configuration space and the high energy barriers. I discuss modifications of these algorithms that partly overcome these problems.

Introduction

Proteins are the nanomachines responsible for nearly every process in our lives, e.g. transporting molecules, catalyzing biochemical reactions and fighting infections.

All proteins are built out of 20 different aminoacids. The unique 3D structure of a naturally occurring protein is determined by its amino acid sequence. These structures are responsible for the function of the protein.
My interest is in the process of folding, especially misfolding events, which cause various diseases. After reconstructing the folding process this knowledge enables the design of new drugs with customized properties.
But proteins consist of many atoms with not wellknown interactions making an investigation difficult. So I have to think about a model for proteins and methods to research it.



Basics of Computional Physics

There are two numerical ways of finding solutions: Molecular Dynamics and Monte Carlo. Molecular Dynamics solves the equations of motion for every single atom step by step numerically, but this is very time consuming because of the many interactions caused by the great number of other atoms. Monte Carlo is used to calculate the statistical properties of the whole system consisting of all atoms, Molecular Dynamics provides statistical properties, too. Such properties are, e.g., energy, radius of gyration, heat capacity and end-to-end distance. I am not interested in the time development of individual properties but in the statistic averages, so I will discuss only the statistical properties in this report.

Basics of Statistical Physics

For ease of notation I will speak about states instead of configurations and expect that the energy has discrete values, which is a simplification, but as I want to implement a model in a computer I need to discretize. If the system interacts with a reservoir at a constant temperature T, in equilibrium every state μ occurs with a certain probability p_μ. Gibbs showed in 1902 that these occupation probabilities are given by:

$$p_\mu(T) = \frac{\mathrm{e}^{-E_\mu/\mathrm{k_B}T}}{\sum_\nu \mathrm{e}^{-E_\nu/\mathrm{k_B}T}},$$

where E_μ is the energy of the state μ and $\mathrm{k_B}$ is the Boltzmann's constant. I denote $1/\mathrm{k_B}T$ with β, which is called the inverse temperature. It holds that

$$p_\mu(T) \propto \mathrm{e}^{-\beta E_\mu},$$

This distribution is called Boltzmann distribution. I can rewrite the distribution of the states into a distribution of the energies

$$P(E,T) = \frac{\Omega(E)\mathrm{e}^{-\beta E}}{\sum_E \Omega(E)\mathrm{e}^{-\beta E}},$$

where $\Omega(E)$ is the density of states counting the number of states with energy E. The distribution remains unchanged if I replace $\Omega(E)$ by $C \cdot \Omega(E)$. If the density of states is known, I obtain the distribution of the energies and therefore the meanvalues of the energy.

If I measure some quantity Q in an experiment repeatedly I can calculate the expectation value of the quantity $\langle Q \rangle$. This value is given as the quantity in a state times the occupation probability of that state, summed over all possible states

$$\langle Q \rangle(T) = \sum_\mu Q_\mu p_\mu(T).$$

Or, if the quantity can explicitly be described in terms of the energy,

$$\langle Q \rangle(T) = \sum_E Q(E) P(E,T).$$

Monte Carlo Simulations

Usually a measurement does not take infinite time and the system will not pass through every state in the sum of expection value. Therefore I can reduce the sum from all possible states to the important states. This procedure is called *importance sampling*.

To produce such a subset of important states I use a Markov Chain process which produces a new state ν out of a given state μ with some transition probability $P(\mu \to \nu)$.

$$\mu \overset{P(\mu\to\nu)}{\longrightarrow} \nu \overset{P(\nu\to\lambda)}{\longrightarrow} \lambda$$

This process has to fulfill some conditions:

- Normalization: $\sum_\nu P(\mu \to \nu) = 1$
  This is necessary to ensure that the transition properties are normalized and that at least one transition is possible.
- Ergodicity:
  It must be possible that every state can be reached from every state in a finite number of steps.

- Detailed Balance: $s_\mu P(\mu \to \nu) = s_\nu P(\nu \to \mu)$
  If this condition is satisfied, the occupation probabilities of the states in the chain are given by s_μ.

From a chain of M measurement of Q I can calculate an estimator of the expection value .

$$\tilde{Q}(T) = \frac{\sum_{i=1}^{M} Q_i s_i^{-1} \mathrm{e}^{-\beta E_i}}{\sum_j s_j^{-1} \mathrm{e}^{-\beta E_j}}$$

For the case of a Boltzmann distributed chain I get

$$\tilde{Q}(T) = \frac{\sum_{i=1}^{M} Q_i}{M}$$

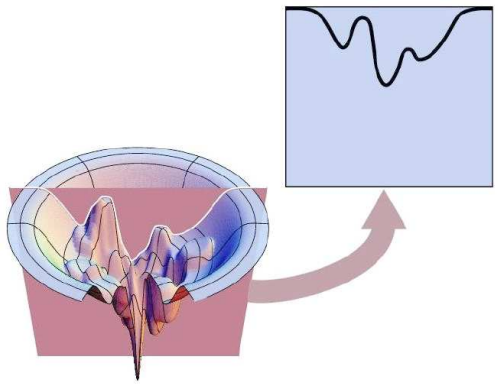
The only open question is the choice of the transition probabilities, I take the choice of Metropolis [1].

$$P(\mu \to \nu) = \min\left(1, \frac{s_\nu}{s_\mu}\right)$$

For $s_\mu = p_\mu$ this is the normal Metropolis Update.

Advanced Monte Carlo

The normal Metropolis Update is inefficient in simulations of systems such as proteins that are characterized by a rough energy landscape. Small changes in configuration can cause great changes in energy. If the system finds a state with an energy smaller than all the states the update could reach in the next step, a so called local minima (see Fig. to the right [2]), the probability to leave a local minima falls off exponentially with the height of the energy barrier. Hence there is a need for better algorithms than the normal Metropolis Update.



Multicanonical Simulations

Here, the idea [3] is to sample all possible energies with nearly equal probability. This helps solving problems caused by big differences in the density of states which can go over 100 orders of magnitude for proteins. The distribution to sample is the multicanonical distribution

$$P_{\text{MuCa}}(E) = W_{\text{MuCa}} \Omega(E) \approx \text{constant}$$

The density $\Omega(E)$ contains all energy information about the system but the density is a priori unknown, so the multicanonical weights $W_{\text{junghans/MuCa}}(E)$ are also unknown and need to be determined. In this algorithm the multicanonical weights are the input to the Metropolis update $s_\mu = W_{\text{junghans/MuCa}}(E_\mu)$.

One way to determine the weights is the multicanonical recursion [4] which is based on some conditions:

- Discrete energies with smallest energy difference ϵ
- Histogram entries $H(E)$ are uncorrelated

I rewrite the weight factor:

$$W_{\text{MuCa}} = \mathrm{e}^{-b(E)E+a(E)}$$

where $b(E)$ is the mircocanonical temperature and $a(E)$ the mircocanonical free energy. These two quantities are linked by the definition of the mircocanonical temperature:

$$a(E-\epsilon) = a(E) + (b(E-\epsilon) - b(E))E.$$

I choose $a(E_{\text{max}}) = 0$. For the beginning I use $b^0(E) = 0 = a^0(E)$ and $g^0(E) = 0$ as normalization parameter. Then the recursion works as follows:

- Determine the nth histogram of the energies $H^n(E)$ with the help of W^n_{MuCa}
- $\kappa^n(E) = \frac{H^n(E+\epsilon)H^n(E)}{H^n(E+\epsilon)+H^n(E)}$
- $g^{n+1}(E) = g^n(E) + \kappa^n(E)$
- $b^{n+1}(E) = b^n(E) + \frac{\kappa^n(E)}{g^{n+1}(E)} \cdot \frac{\ln H^n(E+\epsilon) - \ln H^n(E)}{\epsilon}$

After doing the recursion n_{MuCa} times with l_{MuCa} sweeps, I get a good estimate for the multicanonical weights. The number of runs n_{Muca} is reached if the weights change only very little in comparison to the last run. Then I have to do a final run of m_{MuCa} sweeps with fixed weights in which I measure all interesting quantities.

Wang-Landau Algorithm

All the energy information are given by the density of states $\Omega(E)$. The Wang-Landau algorithm [5] was designed to estimate the density of states directly. This is a different starting point than the idea of multicanonical simulations, but it turns out to be a good input to the multicanonical final run.

First I initialize all weights $W_{\text{WL}}(E_\mu) = 1 = s_\mu$ and the shape parameter $f = \mathrm{e}^1$. Now I perform a Metropolis update and change the weight of the visited energy by $W_{\text{WL}}(E) \to W_{\text{WL}}(E)/f$. After n_{WL} sweeps I change the shape parameter by $f \to \sqrt{f}$ and go on. This change also gives the beginning of the next run. If the shape parameter is approximately 1 the simulation is finished and an estimate for the density of states is given by:

$$\Omega(E) = 1/W_{\text{WL}}(E)$$

This algorithm does not fulfill detailed balance and there is no way of calculating the errors of the density. To solve these problems I using the Wang-Landau weights as input for the multicanonical final run because:

$$W_{\text{MuCa}}(E) \propto 1/\Omega(E) = W_{\text{WL}}(E)$$

Another problem is that the energies have to be discrete, so binning is necessary.

Random Tempering

Simulated Tempering [6] expands the configuration space with an extra temperature coordinate. I perform two kinds of updates:

- Normal Metropolis update:

$$P(E \to E') = \min\left(1, \frac{\mathrm{e}^{-\beta E'}}{\mathrm{e}^{-\beta E}}\right)$$

- Temperature change update:

$$P(\beta \to \beta') = \min\left(1, \frac{\mathrm{e}^{-\beta' E - g(\beta')}}{\mathrm{e}^{-\beta E - g(\beta)}}\right)$$

The system has different equilibration energies at different temperatures, so that the sampled configuration space is much bigger. Without the weights $g(\beta)$ the acceptance ratio of the temperature changes are very small so I introduce these weights, but now I have to determine them first. I choose to do it with the help of the Wang-Landau method.

First I initial all weights $g(E) = 0$ and the shape parameter $h = 1$. Then I perform the updates in alternating order changing the weights of the visited temperature after every cycle by $g(T) \to g(T) + h$. If the histogram of the visited temperatures $H(T)$ becomes sufficiently flat I change the shape parameter by $h = h/2$. Sufficiently flat means the deviation from the mean is at most 20% of the mean. Continue until $h \approx 0$. Because this new method performs a random walk in temperature space I call this enriched version of simulated tempering the random tempering method.

Parallel Tempering

This method [7] is very similar to simulated tempering but works with n copies of the system instead of one. Every system is simulated at a different temperatures and I perform two kind of updates:

- Normal Metropolis update:

$$P(E \to E') = \min\left(1, \frac{\mathrm{e}^{-\beta E'}}{\mathrm{e}^{-\beta E}}\right)$$

- Configuration swap of 2 systems:

$$P(\mu \leftrightarrow \nu) = \min\left(1, \frac{\mathrm{e}^{-\beta_\mu E_\nu - \beta_\nu E_\mu}}{\mathrm{e}^{-\beta_\mu E_\mu - \beta_\nu E_\nu)}}\right)$$

There are no weights to determine. This method has a natural implementation on parallel computers.

Protein Simulations

The methods described above are used in many different fields of computer simulation and also in protein simulation. What makes protein folding so complicated is the rough energy landscape caused by the interactions of the great number of atoms. One simplified description of the interactions is given by the following force field.

Force Field

The energy function [8] I used is measured in kcal/mol:

$$E_{\text{tot}} = E_{\text{LJ}} + E_{\text{el}} + E_{\text{hb}} + E_{\text{tors}}$$

where

- Lennard-Jones term $E_{\text{LJ}} = \sum_{j>i} \left(\frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}}\right)$

- Electrostatic term $E_{\text{el}} = \sum_{i,j} \frac{332 q_i q_j}{\epsilon r_{ij}}$
- Hydrogen-bond term $E_{\text{hb}} = \sum_{j>i} \left(\frac{C_{ij}}{r_{ij}^{12}} - \frac{D_{ij}}{r_{ij}^{10}} \right)$
- Torsion term $E_{\text{tors}} = \sum_l U_l (1 \pm \cos(n_l \chi_l))$

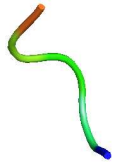
The missing constants are taken from [8]. The Lennard-Jones term takes the radii of the atoms and repulsion of the electron clouds of the atoms into account, the electrostatic term gives the interactions between charged particles, the Hydrogen-bond term describes the energy caused by the polarization of the atom and the torsion term stands for the energy stored in the torsion of the bonds.

Objects of Studies

It is not possible to understand all aspects of folding in one universal protein. I study three different small peptides.

- Met-encephalin

  $$\text{Tyr} - \text{Gly} - \text{Gly} - \text{Phe} - \text{Met}$$

  This is the work horse of algorithms tested in the field of protein folding.

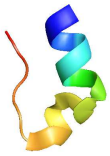
- Alanine 10

  $$\text{Ala} - \text{Ala} - \text{Ala} - \text{Ala} - \text{Ala} - \text{Ala} - \text{Ala} - \text{Ala} - \text{Ala} - \text{Ala}$$

  A littler bigger than Met-encephalin, but still with simple properties.

- Trp-Cage

  $$\text{Asn} - \text{Leu} - \text{Tyr} - \text{Ile} - \text{Gln} - \text{Trp} - \text{Leu} - \text{Lys} - \text{Asp} - \text{Gly} -$$
  $$\text{Gly} - \text{Pro} - \text{Ser} - \text{Ser} - \text{Gly} - \text{Arg} - \text{Pro} - \text{Pro} - \text{Pro} - \text{Ser}$$

  This is a much bigger molecule with nontrivial folding behavior and a characteristic groundstate.

Results

Numerical Comparison of Multicanonical Simulation and Wang-Landau Sampling

As it is the first time that the Wang-Landau algorithm was used together with this force field I first compare the density of states calculated with Wang-Landau with results from a multicanonical simulation. Good analogy can be seen in Figure 1. The graphs are normalized on the energy bin next to the highest energy. For technical reasons the last bin of the multicanonical simulation is filled with all higher energies. Out of this density the meanvalue of the energy was calculated (see Fig. 2) and showed good results at temperatures above room temperature and small deviation at lower temperatures.

The final simulation runs always had a length of $m_{\text{MuCa}} = 100.000$ sweeps. The bin size was 1 kcal/-mol. The interesting energy region was $[-12 \ldots 60]$ kcal/mol for Alanine 10 and $[-12 \ldots 20]$ kcal/mol for Met-encephalin. For weight determination in multicanonical simulation I used 20 recursion run with

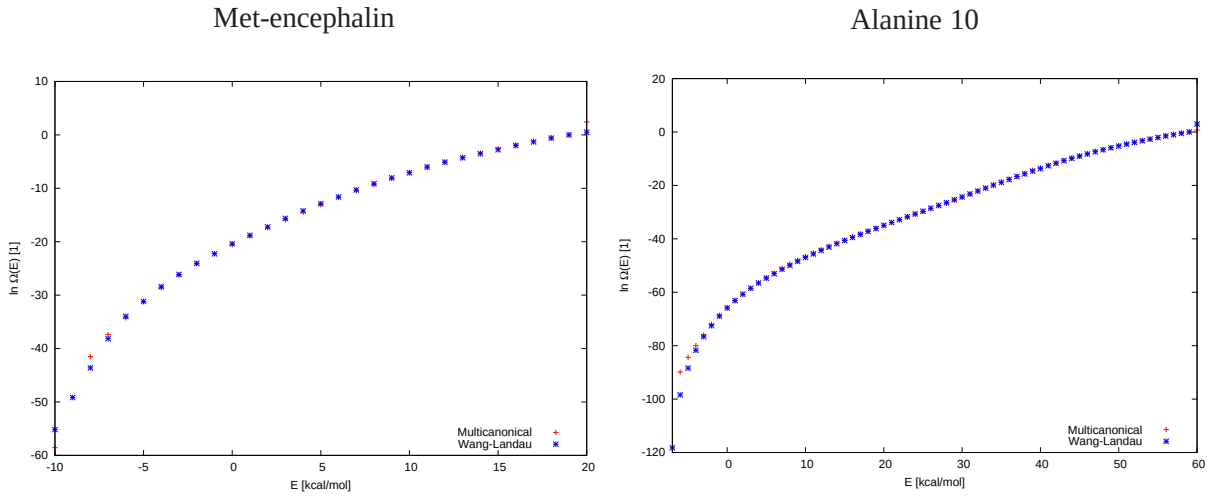


Figure 1: Comparison of the density of states with multicanonic simulation and Wang Landau sampling for Met-encephalin and Alanine 10. Good analogy showed up in both cases.

$n_{\text{Muca}} = 5.000$ sweeps and 10 runs with $n_{\text{WL}} = 10.000$ sweeps for Wang-Landau sampling. Depending on the start configuration the weight finding with Wang-Landau was around 10% faster than the multicanonical recursion.

I also tried using the Wang-Landau algorithm for Trp-cage, but it fails. The problem is the big size of the configuration space. The interesting energy range is $[-170 \dots 50]$ kcal/mol. This caused also a big difference in the order of magnitude of the density of states. The system gets stuck in some state around the ground state and has no chance to move to higher energies again so the meanvalue for small temperatures becomes too small (see Fig. 4). Multicanonical simulations have the same problems here. For big peptides there are too many configurations with the same energy. It is not possible to visit all in finite time. Configuration with similar energy do not have large overlap, which means a big distance in configuration space, so I need to try other methods.

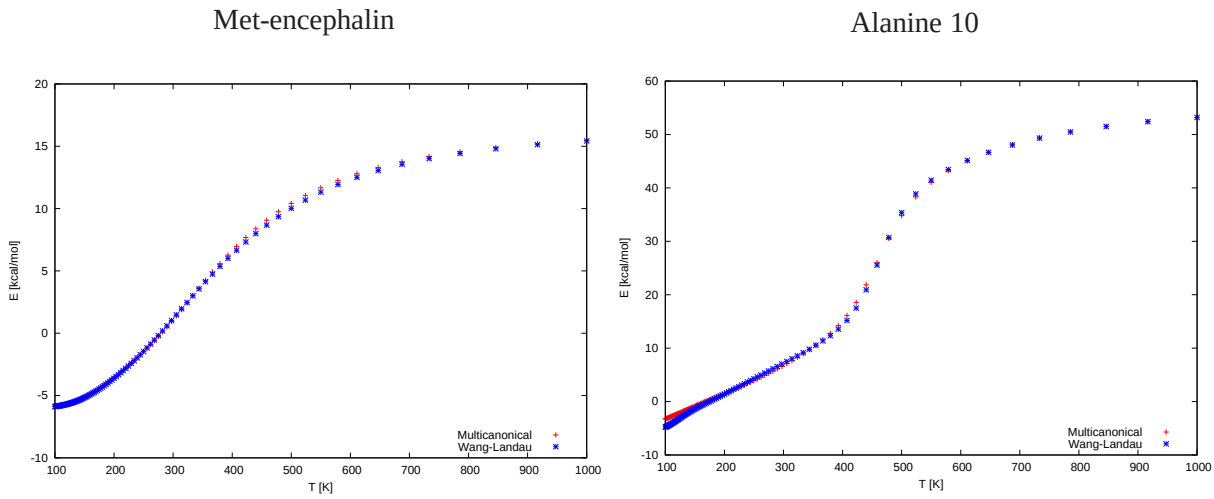


Figure 2: Comparison of the meanvalue of the energies with multicanonical simulation and Wang Landau sampling for Met-encephalin and Alanine 10. Good analogy can be seen for room temperature or higher and small deviation for the lower energy area.

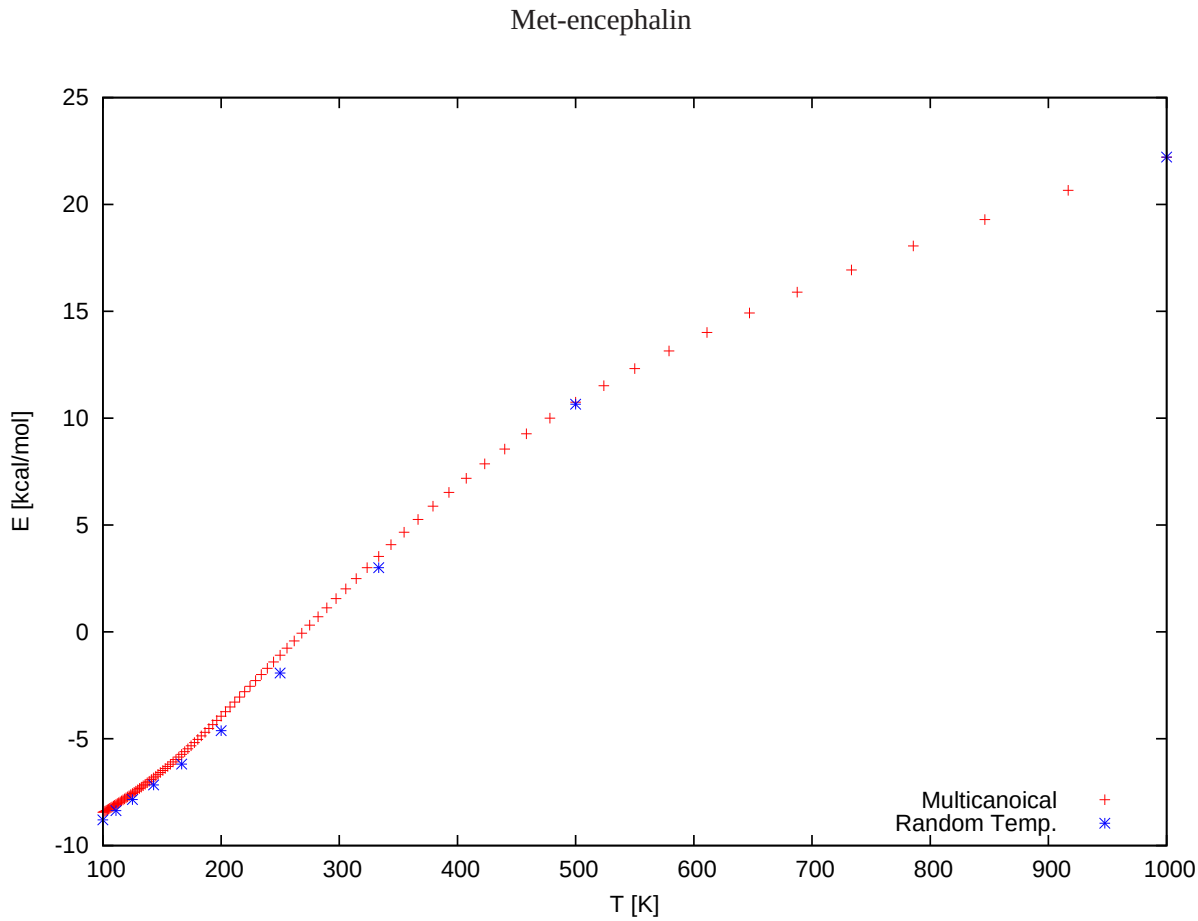


Figure 3: Comparison of multicanonical simulation and random tempering for Met-encephalin. The sampled temperature point of random tempering fit together with the curve of the multicanonical simulation.

Comparison of Random Tempering and Multicanonical Simulation

Multicanonical simulations and Wang-Landau sampling give us information about all temperatures in one simulation. In most cases I am just interested in values at some fixed values of temperature. As in the last section, I first want to test if random tempering works correctly for Met-encephalin. I choose ten temperatures between 100 K and 1000 K. The temperatures are equidistant on the inverse temperature scale because the distribution depends on β not T.

The determination of the weights for random tempering takes around 1.000.000 sweeps in 15 runs to get a sufficiently flat distribution in temperature. It also takes the same number of sweeps to get a flat distribution in the final run with fixed weights. In comparison to multicanonical simulation this is a long time but the results look similar. Results for Met-encephalin are shown in Fig. 3.

For Trp-cage the distribution is not flat even after the 10th run of determining the weights. In the final run the distribution is not flat either and causes some problem in low energy regions (see Fig. 4).

Comparison with Parallel Tempering

As even random tempering has problem with low energy regions, I test parallel tempering for Trp-cage on the supercomputer JUMP in the Research Center Jülich. One time 16 CPUs $\times$ 10 h with 1.100.000 sweeps and the other time 10 CPUs $\times$ 10 h with 1.000.000 sweeps. The results are consistent even with different temperature points (Fig. 4). They have small error bars and are repeatable.

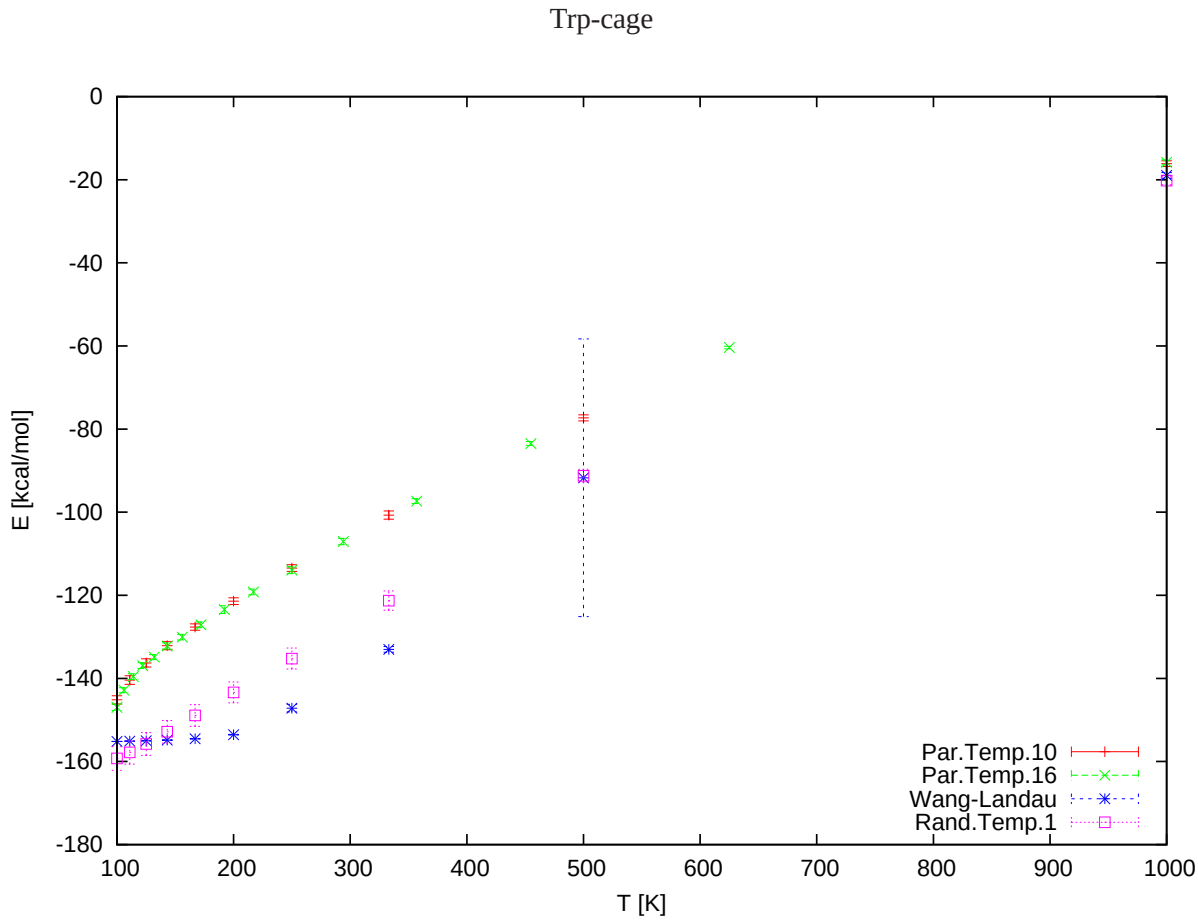


Figure 4: Comparison of several methods for Trp-cage. Random tempering and Wang-Landau sampling have problem in the low energy regions. High temperature points fit together.

Conclusion

All methods I have tested except parallel tempering have produced problems with low temperatures and therefore with the low energy region of big peptides. The statistical results are not repeatable with other starting conditions. This is caused by the size of the configuration space and the large energy barriers. For the smallest tested peptides all methods work well. Maybe it is possible to get better results in a longer simulation run. But in comparison to parallel tempering the efficiency is bad.

Outlook

Because of the very long simulation times it is necessary to parallelize the broad histogram methods like multicanonical simulation and Wang-Landau sampling. This would help to get the same information in shorter time, so that I can have more information about the system in the same time. Till now I cannot even think about simulating a real protein on a supercomputer.

Also I have to think about a way to overcome those configuration barriers. Maybe the choice of the interesting regions in the configuration space have to be done first. Also getting a flat histogram in energy space is not always the best, a flat distribution in overlap to the ground state, which must be experimentally known, may be better.

Acknowledgment

I thank Ulrich Hansmann for giving me this topic near to the current research interests. Also I like to thank Jan Meinke for helping me with all my scientific, technical and administrative problems. Last but not least I thank Thomas Neuhaus for interesting discussions on Advanced Monte Carlo Methods.

References

1. Metropolis et al., Jour. Chem. Phys. 21(1953), 1087-1092
2. Dill and Chan,Natu. Stru. Biol. 1997, 4
3. Berg and Neuhaus, Phys. Rev. Lett. 68(1992), 9
4. Berg, Comp. Phys. Comm. 153(2003), 397-406
5. Wang and Landau, Phys. Rev. E 64(2002), 056101
6. Marinari and Parisi, Europhys. Lett. 19(1992), 451-458
7. Hukushima and Nemoto, Jour. Phys. Soc. Jap. 65(1996), 1604-1608
8. Hansmann et al., Comp. Phys. Comm. 138(2001), 192-212

Multilevel Präkonditionierung für ungeordnete Systeme

Karsten Kahl

Universität Wuppertal

E-mail: kkahl@studs.math.uni-wuppertal.de

Zusammenfassung: Motiviert durch physikalische Probleme in der QED und QCD, bei der große ungeordnete Systeme zu lösen sind, das heißt Systeme bei denen die Kopplungen im Gitter auch einen zufälligen Charakter besitzen, wird versucht, über das Schur-Komplement ein Multilevel Verfahren zu entwerfen. Dieses sollte in der Lage sein, Probleme effizient zu lösen, denen es an genügend geometrischer Information mangelt, um die bekannten Mehrgitter/Multilevel Verfahren anzuwenden. Hierbei wird, wie in traditionellen Multilevel-Verfahren, zunächst eine Vergröberung des zugrundeliegenden Gitters vorgenommen, um dann eine Approximation des Schur-Komplementes als Teil eines Präkonditionierers für ein CG-Verfahren zu verwenden.

Einleitung

Die ungeordneten linearen Probleme, die bei der Diskretisierung von Vorgängen in der QED oder QCD entstehen, sind von großem Interesse, und es ist bislang noch nicht hinreichend gelungen ein Multilevel-Verfahren für diese Probleme zu konstruieren, das allen Anforderungen bezüglich Konvergenz und Effizienz gerecht wird. Neben adaptiven algebraischen Mehrgitteransätzen ist daher die Idee der Schur-Komplement Präkonditionierung entstanden, die ich während meines Aufenthaltes in Jülich untersucht habe. Im Folgenden wird versucht, die grundlegenden Ideen und Ansätze darzustellen, die dafür notwendig sind. Im Rahmen des Praktikums ist auch eine Umsetzung dieses Verfahrens in C/C++ entstanden, und es sind erste Tests durchgeführt worden.
In diesem Bericht werde ich zu Beginn auf das Modellproblem eingehen, mit dem das Verfahren getestet wurde, ehe ich in die Details der Umsetzung gehe. Hier wird, nach der Darstellung des Modellproblems, zunächst die grundlegende Idee der Multilevel Präkonditionierung skizziert, und die auftauchenden Probleme werden hervorgehoben. Dies sind zum einen die Aufteilung des Gitters in grobe und feine Gitterpunkte, wofür dann näher auf die Idee der kompatiblen Relaxation eingegangen wird, und zum anderen die Approximation des Schur-Komplementes mit gewissen erwünschten Eigenschaften, wobei dabei besonderes Augenmerk auf die Ideen des Lumpings und Probings gelegt wird. Abschließend werden noch Ergebnisse der Implementierung dargestellt und ein kleiner Ausblick gegeben, wie man die Idee weiter verfolgen könnte.

Das 2d-$\mathcal{O}(1)$ Modellproblem

Das hier betrachtete 2d-$\mathcal{O}(1)$ Modellproblem wird auf lange Sicht durch interessantere physikalische Modelle, wie das Schwinger Modell der QED oder die Wilson Fermionen-Matrix der QCD, zu ersetzen sein.

Definition 1. *Sei eine Gittergröße $N \in \mathbb{N}$ gegeben und ein Temperaturparameter $\beta \in \mathbb{R}$, so ist das 2d-$\mathcal{O}(1)$ Modell Problem ein lineares System mit den Variablen u_{ij}, die zum äquidistanten Gitter $\Omega_N = \{(i,j) \mid i,j = 1,\dots,N1\}$ gehören. Das System wird beschrieben durch den periodischen 5-Punkte-Stern gegeben durch*

$$u_{ij} - \kappa(e^{2\pi i\beta\theta_{ij}}u_{i-1,j} + e^{2\pi i\beta\varphi_{ij}}u_{i,j-1} + e^{-2\pi i\beta\theta_{i+1,j}}u_{i+1,j} + e^{-2\pi i\beta\varphi_{i,j+1}}u_{i,j+1}) = f_{ij}$$

Die Variablen $\theta_{ij}, \varphi_{ij} \in \mathbb{R}$ sind unabhängige Ausprägungen von $N(0,1)$-normalverteilten Zufallszahlen. Der Parameter $\kappa > 0$ wird Kopplungsparameter genannt.

Bemerkung 1.

- *Der Name des Modellproblems veranschaulicht, dass es sich um ein 2-dimensionales Gitter handelt und die Kopplungskoeffizienten aus der Gruppe $\mathcal{O}(1)$ der eindimensionalen orthogonalen Matrizen stammen, also komplexe Zahlen mit Modulus 1 sind.*

- *Das lineare System ist hermitesch, da die Kopplung von Knoten (i,j) zu Knoten (i+1,j) $\alpha = \kappa e^{2\pi i\beta\theta_{ij}}$ die konjugiert komplexe Zahl zur Kopplung $\gamma = \kappa e^{-2\pi i\beta\theta_{ij}}$ des Knotens (i+1,j) zum Knoten (i,j) ist. Analog für die andere Dimension.*

- *Der Kopplungsparameter κ wird immer so gewählt, dass die Systemmatrix positiv definit ist und der kleinste Eigenwert der Matrix in der Größenordnung von $\frac{1}{N^2}$ liegt, hierzu wird $\kappa \in [0.2, 0.4]$ liegen.*

- *Erhöht man den Wert von $\beta > 0$ so erhöht man den Zufall im System. Für $\beta = 0,\ \kappa = 0.25$ erhält man das periodische Laplace-System, in unserem Modellproblem sind Werte $\beta \in [0.2, 0.6]$ von Interesse.*

Alle Kopplungen in diesem System besitzen die gleiche Stärke, sodass eine Unterscheidung nach nicht möglich ist. Um jedoch aus der Menge der Gitterpunkte sinnvoll ein grobes Gitter auswählen zu können, ist eine Unterscheidbarkeit notwendig. Dies motiviert den Übergang zum odd-even-reduzierten System.

Odd-even Reduzierung

Definition 2. *Ein Knoten heißt **even**, wenn $(i+j)mod\ 2 = 0$ und **odd**, wenn $(i+j)mod\ 2 = 1$ ist.*

Beachtet man, dass im 2d-$\mathcal{O}(1)$ Modell-Problem jeder even (bzw. odd) Knoten nur mit seinen vier benachbarten odd (bzw. even) Knoten koppelt, erhält man durch Permutation

$$A = \begin{pmatrix} I & D_{oe} \\ D_{eo} & I \end{pmatrix},\ D_{eo} = D_{oe}^H$$

Hieraus lässt sich das System auf die even-Knoten reduzieren, indem man

$$A_{ee} = I - D_{eo}D_{oe}$$

betrachtet. Beachte, dass das ursprüngliche System

$$A \cdot \begin{pmatrix} x_o \\ x_e \end{pmatrix} = \begin{pmatrix} b_o \\ b_e \end{pmatrix}$$

nun gelöst wird durch $A_{ee}x_e = b_e - D_{eo}b_o,\ x_o = b_o - D_{oe}x_e$. Die Matrix A_{ee} des reduzierten Systems kann verstanden werden als ein 9-Punkte-Stern auf den even-Knoten. A_{ee} ist besser konditioniert als die Ausgangsmatrix A.

Lemma 1. *Sei λ ein Eigenwert von A und $\lambda_1 > 0$ bezeichne den kleinsten Eigenwert von A. Dann gilt:*

i. $2 - \lambda$ *ist auch ein Eigenwert von* A

ii. $cond(A) = (2 - \lambda_1)/\lambda_1$

iii. Für alle Eigenwert μ *von* A_{ee} *gilt* $\mu = \lambda \cdot (2 - \lambda)$, λ *Eigenwert von* A

iv. A_{ee} *ist besser konditioniert als* A*, es gilt*

$$cond(A_{ee}) \leq \frac{1}{\lambda_1(2-\lambda_1)} = \rho \cdot cond(A), \text{ mit } \rho = \frac{1}{(2-\lambda_1)^2} \approx \frac{1}{4}$$

$Beweis$:

i. Aus
$$A \cdot \begin{pmatrix} x_o \\ x_e \end{pmatrix} = \lambda \cdot \begin{pmatrix} x_o \\ x_e \end{pmatrix}$$
folgt mit ein paar leichten Umformungen
$$A \cdot \begin{pmatrix} x_o \\ -x_e \end{pmatrix} = (2-\lambda) \cdot \begin{pmatrix} x_o \\ -x_e \end{pmatrix}$$

ii. Folgt aus i.

iii. Man erhält aus
$$A \cdot \begin{pmatrix} x_o \\ x_e \end{pmatrix} = \lambda \cdot \begin{pmatrix} x_o \\ x_e \end{pmatrix}$$
sofort
$$(I - D_{eo}D_{oe})x_e = \lambda(2-\lambda)x_e$$
Also ist $\lambda(2-\lambda) \in spec(Aee)$. Desweiteren kann die Matrix $A_{ee} = I - D_{eo}D_{oe}$ keinen Eigenwert größer als 1 besitzen, da $D_{eo} = D_{oe}^H$ gilt. Damit hat die Gleichung $\mu = \lambda \cdot (2-\lambda)$ eine (im Fall $\mu = 1$) oder zwei reelle Lösungen (im Fall $\mu < 1$) für λ. Ist nun x_e Eigenvektor zu $\mu < 1$, $\lambda(2-\lambda) = \mu$ und $x_o = \frac{1}{\lambda - 1} D_{oe} x_e$, so ist
$$A \begin{pmatrix} x_o \\ x_e \end{pmatrix} = \lambda \begin{pmatrix} x_o \\ x_e \end{pmatrix}$$
Genauso im Fall $\mu = 1$ mit zugehörigem Eigenvektor x_e und $x_o = 0$ folgt
$$A \begin{pmatrix} 0 \\ x_e \end{pmatrix} = \begin{pmatrix} 0 \\ x_e \end{pmatrix}$$
also gilt $spec(A_{ee} = \{\lambda(2-\lambda),\ \lambda \in spec(A)\}$.

iv. Da $\lambda(2-\lambda) \leq 1$ für alle $\lambda \in [\lambda_1, 2-\lambda_1]$ gilt und für $\lambda = \lambda_1$ sein Minimum annimmt, folgt $iv.$ aus $iii.$.

Das odd-even reduzierte System entspricht einem 9-Punkte-Stern auf den even Knoten, sodass sich für jeden Knoten (i,j) folgende Gleichung ergibt

$$\begin{array}{ll}
(1-4\kappa^2) \cdot u_{ij} - \kappa^2 \cdot ((e^{2\pi i\beta(\theta_{ij}+\varphi_{i-1,j})} & +e^{2\pi i\beta(\theta_{i,j-1}+\varphi_{ij})}) \cdot u_{i-1,j-1} \\
+(e^{2\pi i\beta(\theta_{ij}-\varphi_{i-1,j+1})} & +e^{2\pi i\beta(\theta_{i,j+1}-\varphi_{i,j+1})}) \cdot u_{i-1,j+1} \\
+(e^{2\pi i\beta(-\theta_{i+1,j}+\varphi_{i+1,j})} & +e^{2\pi i\beta(-\theta_{i+1,j-1}+\varphi_{ij})}) \cdot u_{i+1,j-1} \\
+(e^{2\pi i\beta(-\theta_{i+1,j}-\varphi_{i+1,j+1})} & +e^{2\pi i\beta(-\theta_{i+1,j+1}-\varphi_{i,j+1})}) \cdot u_{i+1,j+1} \\
+e^{2\pi i\beta(\theta_{ij}+\theta_{i-1,j})} \cdot u_{i-2,j} & +e^{2\pi i\beta(\varphi_{ij}+\varphi_{i,j-1})} \cdot u_{i,j-2} \\
+e^{2\pi i\beta(-\theta_{i+2,j}-\theta_{i+1,j})} \cdot u_{i+2,j} & +e^{2\pi i\beta(-\varphi_{i,j+2}-\varphi_{i,j+1})} \cdot u_{i,j+2}) = f_{ij}
\end{array}$$

Man sieht, dass die Kopplungskoeffizienten sich nun unterscheiden sollten, da die Kopplungen von (i,j) nach $(i \pm 2, j)$ und $(i, j \pm 2)$ die Grössenordnung κ^2 besitzen und die Größe der übrigen Kopplungen größer zu erwarten ist, wie folgendes Lemma zeigt.

Lemma 2. *Seien θ und φ zwei unabhängige $N(0,1)$-normalverteilte Zufallszahlen, dann gilt* $\langle |e^{2\pi i\theta} + e^{2\pi i\varphi}| \rangle = \sqrt{2 + 4e^{-8\pi^2}}$

$Beweis$:

$$\langle |e^{2\pi i\theta} + e^{2\pi i\varphi}|^2 \rangle = \frac{1}{2\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (2 + e^{2\pi i(\theta-\varphi)} + e^{2\pi i(\theta+\varphi)}) \cdot e^{-\theta^2/2} \cdot e^{-\varphi^2/2} d\theta d\varphi$$

Aus der Integralrechnung ist bekannt

$$\int_{-\infty}^{\infty} e^{-at^2} cos(bt) dt = \frac{\sqrt{\pi}}{a} e^{-b^2/(4a^2)}$$

$$\int_{-\infty}^{\infty} e^{-at^2} sin(bt) dt = 0$$

sodass man nun folgendes erhält

$$\begin{aligned}
&\langle |e^{2\pi i\theta} + e^{2\pi i\varphi}|^2 \rangle \\
&= \tfrac{1}{\pi} \textstyle\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (1 + cos(2\pi\theta)cos(2\pi\varphi) + sin(2\pi\theta)(2\pi\varphi)) \cdot e^{-\theta^2/2} \cdot e^{-\varphi^2/2} d\theta d\varphi \\
&= \tfrac{1}{\pi}(2\pi + 4\pi e^{-8\pi^2}) \\
&= 2 + e^{-8\pi^2}
\end{aligned}$$

Obwohl die im Lemma gewonnenen Ergebnisse nicht exakt auf die Kopplungen von Knoten (i,j) zu $(i \pm 1, j \pm 1)$ in dem odd-even reduzierten System anwendbar sind, ist klar, dass sich diese Kopplungen in ihrer Größe von den Kopplungen (i,j) zu $(i \pm 2, j)$ (bzw. zu $(i, j \pm 2)$) unterscheiden.
Da es sich hierbei nur um Erwartungswerte handelt, ist es auch möglich, dass die betrachtete Kopplung des Knotens (i,j) zu $(i \pm 1, j \pm 1)$ sehr schwach ausfällt oder gar ganz verschwindet.

Schur-Komplement Ansatz

Um die grundlegende Idee des Schur-Komplement Präkonditionierers zu erläutern, betrachten wir zunächst einen verallgemeinerten Ansatz, ehe wir zu unserem 2d-$\mathcal{O}(1)$ Modellproblem zurückkehren.
Sei also $A \in \mathbb{C}^{n \times n}$ hermitesch und positiv definit, weiterhin sei eine Aufteilung von A in zwei Blöcke der Größe n_f und n_c gegeben, sodass $n_f + n_c = n$

$$A = \begin{pmatrix} A_{ff} & A_{fc} \\ A_{cf} & A_{cc} \end{pmatrix},\ A_{ff} \in \mathbb{C}^{n_f \times n_f},\ A_{fc} = A_{cf}^H \in \mathbb{C}^{n_f \times n_c},\ A_{cc} \in \mathbb{C}^{n_c \times n_c}$$

A lässt sich faktorisieren zu:

$$A = \begin{pmatrix} A_{ff} & A_{fc} \\ A_{cf} & A_{cc} \end{pmatrix} = \begin{pmatrix} I & 0 \\ A_{cf}A_{ff}^{-1} & I \end{pmatrix} \cdot \begin{pmatrix} A_{ff} & 0 \\ 0 & S \end{pmatrix} \cdot \begin{pmatrix} I & A_{ff}^{-1}A_{fc} \\ 0 & I \end{pmatrix},$$

wobei $S = A_{cc} - A_{cf}A_{ff}^{-1}A_{fc}$ das *Schur-Komplement* ist. Daraus folgt:

$$A^{-1} = \begin{pmatrix} I & -A_{ff}^{-1}A_{fc} \\ 0 & I \end{pmatrix} \cdot \begin{pmatrix} A_{ff}^{-1} & 0 \\ 0 & S^{-1} \end{pmatrix} \cdot \begin{pmatrix} I & 0 \\ -A_{cf}A_{ff}^{-1} & I \end{pmatrix} \cdot$$

Damit lässt sich das System $Ax = b$ bei blockweiser Zerlegung der Vektoren in vier Schritten lösen

$$\begin{array}{ll} (1)\ y_c = A_{ff}^{-1} b_f & (3)\ x_c = S^{-1} y_c \\ (2)\ y_c = b_c - A_{cf} y_f & (4)\ x_f = b_f - A_{ff}^{-1} A_{fc} x_c \end{array}$$

Unter der Annahme, dass Systeme mit A_{ff} einfach zu lösen sein werden, da A_{ff} sich als gut konditioniert herausstellen wird und damit wenige Schritte eines CG-Verfahrens oder einer Relaxation ausreichen, haben wir uns vornehmlich um S zu kümmern. Das Schur-Komplement S ist i.A. dicht besetzt und schlecht konditioniert, sodass wir zu einer Approximation $\tilde{S}$ von S übergehen, die dünn besetzt ist, um dann obige 4 Schritt-Formel zur Lösung von $Ax = b$ als Präkonditionierer in einem CG-Verfahren einzusetzen. An dieser Stelle werden wir nur das 2-Level-Schema betrachten. Der Übergang zum Multi-Level-Schema findet statt, indem man das Lösen von $\tilde{S} x_c = y_c$ wiederum mit einem präkonditionierten CG-Verfahren macht, wobei die Präkonditionierung wie oben beschrieben stattfindet, man also ein zum Schur-Komplement des approximierten Schur-Komplementes übergeht. Der Präkonditionierer kann also in faktorisierter Form geschrieben werden als

$$\begin{aligned} M &= \begin{pmatrix} I & 0 \\ A_{cf} A_{ff}^{-1} & I \end{pmatrix} \cdot \begin{pmatrix} A_{ff} & 0 \\ 0 & \tilde{S} \end{pmatrix} \cdot \begin{pmatrix} I & A_{ff}^{-1} A_{fc} \\ 0 & I \end{pmatrix} \\ M^{-1} &= \begin{pmatrix} I & -A_{ff}^{-1} A_{fc} \\ 0 & I \end{pmatrix} \cdot \begin{pmatrix} A_{ff}^{-1} & 0 \\ 0 & \tilde{S}^{-1} \end{pmatrix} \cdot \begin{pmatrix} I & 0 \\ -A_{cf} A_{ff}^{-1} & I \end{pmatrix} \end{aligned}$$

Damit der Präkonditionierer effektiv ist, sollte $\tilde{S}$ *spektral äquivalent* zu S sein.

Definition 3. *Zwei Matrizen $S, \tilde{S} \in \mathbb{C}^{n \times n}$ beide hermitesch und positiv definit, sind spektral äquivalent, wenn es Konstanten $\gamma, \Gamma \in \mathbb{R}$ mit $0 < \gamma \leq 1 \leq \Gamma$ gibt, sodass*

$$\gamma \tilde{S} \preceq S \preceq \Gamma \tilde{S},$$

wobei mit $\preceq$ die Ordnungsrelation bezüglich des Kegels der hermiteschen, positiv definiten Matrizen gemeint ist.

Sind $\tilde{S}$ und S spektral äquivalent, so ist

$$spec(\tilde{S}^{-1} S) \subseteq [\gamma, \Gamma]$$

sodass für die Konditionszahl gilt

$$cond(\tilde{S}^{-1} S) \leq \Gamma / \gamma.$$

Spektrale Äquivalenz kann also als Beschränkung der Kondition des präkonditionierten Systems $\tilde{S}^{-1} S$ angesehen werden.

Lemma 3. *Es sei $\gamma \tilde{S} \preceq S \preceq \Gamma \tilde{S}$, dann erfüllt die Präkonditionierungsmatrix M*

$$\gamma M \preceq A \preceq \Gamma M$$

$Beweis$: Es ist

$$\Gamma M - A = \begin{pmatrix} I & 0 \\ A_{cf} A_{ff}^{-1} & I \end{pmatrix} \cdot \begin{pmatrix} (\Gamma - 1) A_{ff} & 0 \\ 0 & \Gamma \tilde{S} - S \end{pmatrix} \cdot \begin{pmatrix} I & A_{ff}^{-1} A_{fc} \\ 0 & I \end{pmatrix}$$

hermitesch und positiv definit, da $\Gamma \geq 1$. Also $A \preceq \Gamma M$. Die andere Ungleichung folgt analog.

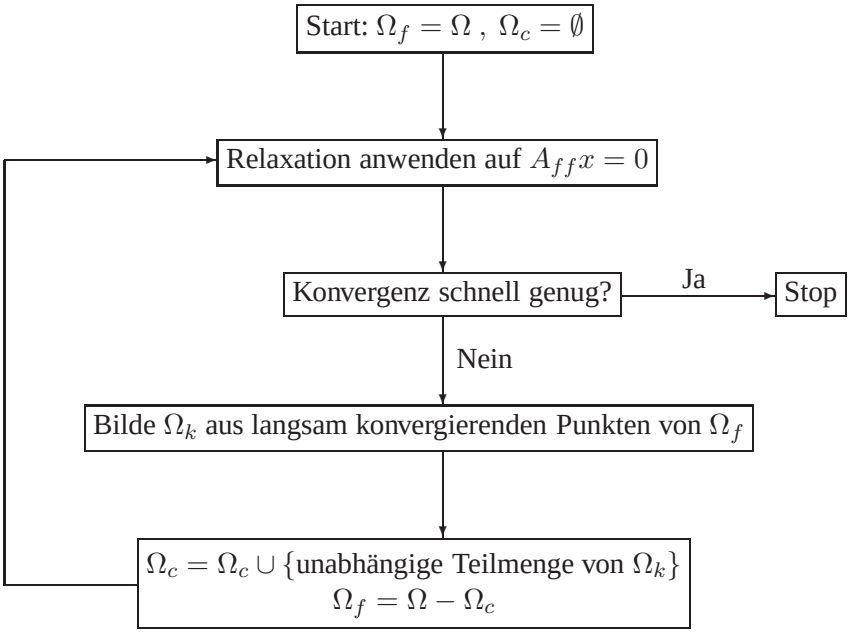
Daraus ergeben sich folgende Forderungen an unser Verfahren:

- Es muss eine Aufteilung des Gitters Ω in ein grobes Ω_c und ein feines Ω_f Gitter gefunden werden, sodass A_{ff} möglichst gut konditioniert und $|\Omega_c|$ möglichst klein ausfällt.
- Das Schur-Komplement S ist durch $\tilde{S}$ zu approximieren, wobei $\tilde{S}$ die Eigenschaften
  - $\tilde{S}$ ist dünn besetzt
  - $\tilde{S}$ ist hermitesch und positiv definit
  - $cond(\tilde{S}^{-1}S) \leq const.$, ($S, \tilde{S}$ sind spektral äquivalent, der Präkonditionierer also effektiv)

  erfüllt sind.

Kompatible Relaxation (Compatible Relaxation, CR)

In diesem Abschnitt beschäftigen wir uns nun mit der Aufteilung des Gitters in feine und grobe Punkte. Genauer um eine Partition der Knotenmenge Ω in eine Menge Ω_f der feinen und Ω_c der groben Gitterpunkte, wobei $\Omega_f \cup \Omega_c = \Omega$ gilt. Wie in der Zusammenfassung erwähnt sind wir interessiert daran die Punkte so aufzuteilen, dass die Matrix A_{ff} gut konditioniert ist, also Systeme der Bauart $A_{ff}x = b$ leicht und effizient zu lösen sind. Um die Kompatible Relaxation besser erläutern zu können, hier zunächst eine Skizze des Algorithmus.



Bemerkung 2. *Man erkennt, dass es sich bei der Kompatiblen Relaxation um ein adaptives Verfahren zur Gewinnung der Aufteilung des Gitters handelt. Die einzelnen Schritte des Algorithmus werden im Folgenden näher beschrieben.*

- *Die verwendete Relaxation war in diesem Fall das Jacobi Verfahren*

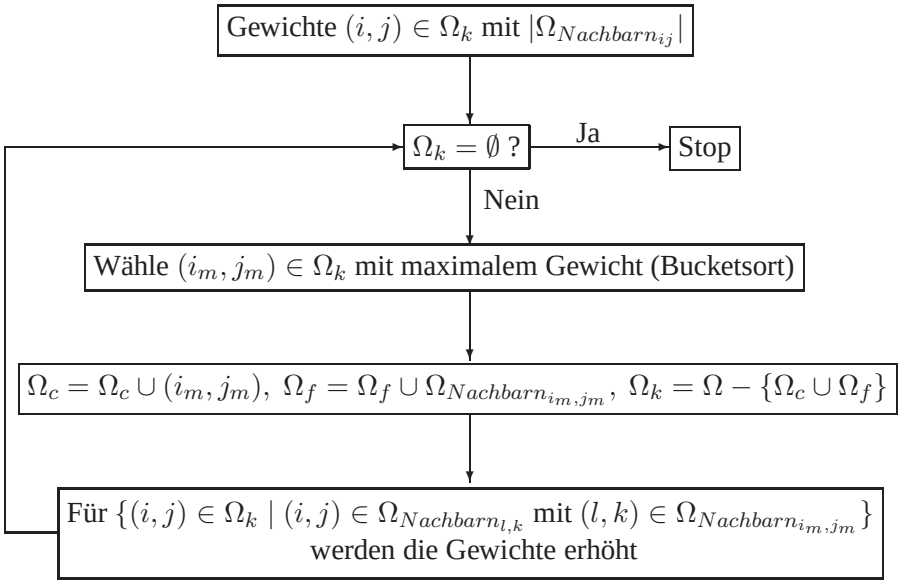
  $$x^{(k+1)} = D^{-1}(I + B)x^{(k)},\ A = D - B,\ D = diag(A)$$

  an dieser Stelle könnte man jedoch auch Verfahren wie Gauss-Seidel oder SSOR einsetzen.
- *Die Konvergenzgeschwindigkeit α wird gemessen durch $\alpha = \frac{\|r^{(k+1)}\|}{\|r^{(k)}\|},\ r^{(k)} = Ax^{(k)},\ k \in \mathbb{N}$. Das Abbruchkriterium wird nach einigen Iterationen geprüft, wenn sich die asymptotische Konvergenzgeschwindigkeit eingestellt hat (Typische Abbruchbedingungen sind $\alpha \in [0.5, 0.75]$).*

- *Knoten werden immer dann zu Kandidaten für grobe Knoten, wenn die Konvergenz der Relaxation auf diesen Knoten unterdurchschnittlich, also die Iterierte $|x_{ij}^{(k)}| > 1 - \alpha$ war. Damit ergibt sich* $\Omega_k = \{(i,j) \in \Omega_f \mid |x_{ij}^{(k)}| > 1 - \alpha\}$

Definition 4. *Punkte $(i_1, j_1), (i_2, j_2) \in \Omega$ heißen benachbart genau dann, wenn $a_{(i_1,j_1),(i_2,j_2)} \neq 0$. Demnach definiert sich die Menge der Nachbarn des Knotens (i,j)*
$\Omega_{Nachbarn_{ij}} = \{(l,k) \in \Omega \mid (l,k)$ *und* (i,j) *sind benachbart* $\}$.

Mit Hilfe dieser Definition lässt sich nun die Auswahl der groben Gitterpunkte aus der Menge der Kandidaten einfacher beschreiben. Auch hier zunächst eine schematische Darstellung.



Es werden also immer die Kandidaten-Knoten zu Knoten des groben Gitters gemacht, die besonders viele benachbarte Kandidaten besitzen. Das Gewicht eines Knotens wird genau dann erhöht, wenn sich in seiner Nachbarschaft ein Knoten befindet, dessen Zustand infolge der Auswahl eines groben Punktes von Kandidat zu fein wechselt. Die Auswahl möglichst unabhängiger Teilmengen aus Ω_k für die Bildung von Ω_c zielt darauf ab, dass die Matrix A_{cc} möglichst dünn besetzt ist.

Nach Terminierung der Kompatiblen Relaxation liegt nun eine Aufteilung der Knotenmenge Ω in Ω_f und Ω_c mit $\Omega_f \cup \Omega_c = \Omega$ vor, die folgende Eigenschaften erfüllt.

- Systeme der Bauart $A_{ff}x = b$ sind einfach zu lösen, da man mit Hilfe d er ausgewählten Relaxation eine asymptotische Konvergenz $\alpha < \alpha_0 \in [0.5, 0.75]$ erreicht wird.
- A_{cc} ist möglichst dünn besetzt.

Schur-Komplement Approximation

In diesem Abschnitt beschäftigen wir uns nun mit dem verbleibenden Problem der Approximation $\tilde{S}$ des Schur-Komplementes $S = A_{cc} - A_{cf}A_{ff}^{-1}A_{fc}$, wobei nach Möglichkeit die Eigenschaften

- $\tilde{S}$ ist dünn besetzt
- $\tilde{S}$ ist hermitesch und positiv definit

- $cond(\tilde{S}^{-1}S) \leq const.$, ($S, \tilde{S}$ sind spektral äquivalent, der Präkonditionierer also effektiv)

erfüllt werden.
Das Problem der dichten Besetzung des Schur Komplementes lässt sich leicht auf die i.A. dichte Besetzung der Inversen A_{ff}^{-1} zurückführen, sodass wir uns zunächst um die Approximation dieser Inversen mit einer erwünscht dünnen Besetzung kümmern. Hierzu, losgelöst von unserem Modellproblem, nun zunächst ein bisschen Theorie aus der linearen Algebra.

Lemma 4. *Sei $A \in \mathbb{C}^{n\times n}$ mit $\|A\| < 1$, dann gilt:*

$$(I - A)^{-1} = \sum_{k=0}^{\infty} A^k$$

$Beweis$: Da $\|A\| < 1$, gilt $A^n \overset{n\to\infty}{\longrightarrow} 0$. Betrachte nun

$$\begin{array}{l}(I - A) \cdot \sum_{k=0}^{n} A^k = I - A^{n+1} \overset{n\to\infty}{\longrightarrow} I \\ \Rightarrow (I - A)^{-1} = \sum_{k=0}^{\infty} A^k\end{array}$$

Diese Darstellung von $(I - A)^{-1}$ wird auch *von Neumann-Reihe* genannt.

Lemma 5. *Sei $A \in \mathbb{C}^{n\times n},\ A = D - B$ mit $D = diagA$ und $\|D^{-1}B\| < 1$ dann gilt:*

$$A^{-1} = (\sum_{k=0}^{\infty}(I - D^{-1}B)^k) \cdot D^{-1}$$

$Beweis$:

$$\begin{array}{l}A = D - B \\ \Rightarrow D^{-1}A = I - D^{-1}B \\ \Rightarrow A^{-1}D = (I - D^{-1}B)^{-1} \\ \overset{\|D^{-1}B\|<1}{\Rightarrow} A^{-1} = (\sum_{k=0}^{\infty}(I - D^{-1}B)^k) \cdot D^{-1}\end{array}$$

Um also ein gutes Besetzungsmuster für die Approximation von $S = A_{cc} - A_{cf}A_{ff}^{-1}A_{fc}$ zu erhalten verwenden wir die Approximation $\tilde{A}_{ff}^{-1} = D_{ff}^{-1} - D_{ff}^{-1}B_{ff}D_{ff}^{-1}$ von A_{ff}^{-1}, da dann
$struct(\tilde{A}_{ff}^{-1}) = struct(A_{ff})$ gilt und die daraus resultierende Approximation $\tilde{S} = A_{cc} - A_{cf}\tilde{A}_{ff}^{-1}A_{fc}$ sowohl eine annehmbare Besetzung aufweist, als auch positiv definit und hermitesch ist.
Eine weitere Verbesserung der Besetzungsstruktur wird noch durch einen *Lumping*-Ansatz versucht, indem wir unwichtige (d.h. sehr kleine) Einträge, die immer als Kopplungsstärken zu verstehen sind, besonders behandeln. Dabei ist zu gewährleisten, dass die zweite Forderung an die Eigenschaften der Approximation erfüllt bleiben. Die dritte Eigenschaft, die maßgeblich für die Güte des Präkonditionierers verantwortlich ist, wird durch diese Approximation i.A. nicht erfüllt, sodass wir versuchen mit einer nachträglichen Anpassung der Approximation über einen *Probing*-Ansatz diese Eigenschaft zu verbessern.

Lumping

Insbesondere in tieferen Leveln des Verfahrens fällt auf, dass die Einträge der Problemmatrix zunehmend an Größe relativ zu den Einträgen auf der Diagonalen verlieren, sodass die Idee auf der Hand liegt, diese Einträge zu vernachlässigen, um die Besetzung der Matrix weiter zu verbessern. Hierbei ist darauf zu achten, dass die positive Definitheit und Hermiteschheit der Matrix nicht verletzt werden. Sei also $\delta \in \mathbb{R}$

die Größe die unterschritten werden muss, damit eine Kopplung als vernachlässigbar eingestuft wird (relativ zur Größe der Einträge auf der Diagonalen wurden Werte $\delta \in [10^{-4}, 10^{-3}]$ getestet). Betrachte dann zur Problemmatrix $A \in \mathbb{C}^{n\times n} : \Delta A = \{a_{ij} \in A \mid |a_{ij}| < \delta\}$ und modifiziere A wie folgt

$$\tilde{A} = A - \Delta A + diag(|\Delta A|(1, \ldots, 1)^T)$$

$$\begin{pmatrix} \alpha & \beta \\ \bar{\beta} & \delta \end{pmatrix} \longrightarrow \begin{pmatrix} \alpha + |\beta| & 0 \\ 0 & \delta + |\beta| \end{pmatrix}$$

Man sieht leicht, dass $\tilde{A} - A$ positiv semi-definit ist und damit auch $\tilde{A} = A + (\tilde{A} - A)$, wenn A positiv definit ist. Durch diese Vorgehensweise erreicht man es, dass die Besetzung der Matrix verbessert wird und die Eigenschaften der Problemmatrix erhalten bleiben. Jedoch ist zu befürchten, dass sich die Güte der Approximation als Präkonditionierer weiter verschlechtert, die Konvergenz in tieferen Leveln des Verfahrens durch das Lumping sogar gänzlich verloren geht, sodass man diesen Ansatz mit Vorsicht verwenden sollte.

Probing

Um die dritte gewünschte Eigenschaft $cond(\tilde{S}^{-1}S) < const.$ zu erhalten, versuchen wir die Approximation $\tilde{S}$ besser an die kleinen Eigenwerte/Eigenvektoren von S anzupassen, da diese zum einen i.A. von unserer bisherigen Approximation nur schlecht wiedergegeben werden und zum anderen maßgeblich für die Güte unseres Multilevel Präkonditionierers verantwortlich sind. Um dies zu erreichen, verwenden wir folgenden Ansatz:

i. Bestimme einen (oder mehrere) typische Vertreter e_{klein} von Eigenvektoren zu kleinen Eigenwerten

ii. Passe $\tilde{S}$ so an, dass $S \cdot e_{klein} \approx \tilde{S} \cdot e_{klein}$

zu $i.$:
Um an typische Vertreter von Eigenvektoren zu kleinen Eigenwerten zu gelangen verwenden wir eine Iteration der Gestalt $e_{klein}^{(k+1)} = (I - \alpha S)e_{klein}^{(k)}$, wobei $e_{klein}^{(0)}$ mit zufälligen Einträgen gewählt wird und $\alpha \in \mathbb{R}$ so bestimmt, dass die Iteration die Anteile im Vektor zu großen Eigenwerten stärker dämpft als zu kleinen. Zum Beispiel wählt man $\alpha = \frac{5}{4\lambda_{max}}$, wobei λ_{max} der größte Eigenwert von S ist.
zu $ii.$:
Setze die neue Approximation $\tilde{\tilde{S}}$ an als $\tilde{\tilde{S}} = A_{cc} - A_{cf}(\tilde{A}_{ff}^{-1} + V_{ff})A_{fc}$, wobei V_{ff} das gleiche Besetzungsmuster hat wie $\tilde{A}_{ff}^{-1}$, also $struct(V_{ff}) = struct(A_{ff})$. Daraus folgt, da die Matrix A_{cf} maximalen Rang besitzt, dass sich die Anpassung reduzieren lässt auf:

$$\underbrace{(A_{ff}^{-1} - \tilde{A}_{ff}^{-1})A_{fc}e_{klein}}_{y} \approx V_{ff} \underbrace{A_{fc}e_{klein}}_{\tilde{e}_{klein}}$$

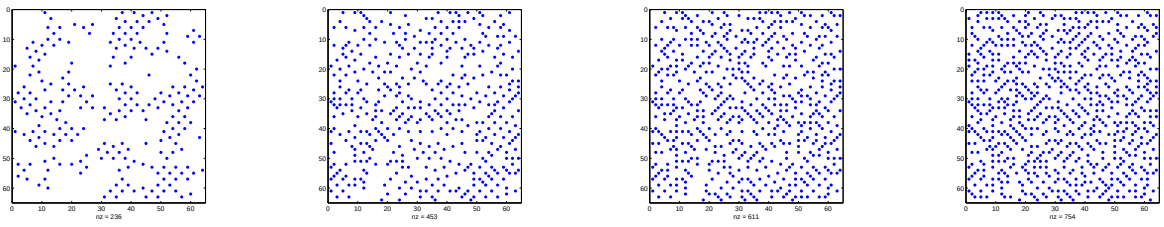
Es ergibt sich also für jede Zeile ein unterbestimmtes Gleichungssystem $y_i = \tilde{e}_{klein} \cdot (v_{ij_1}, \ldots, v_{ij_m})^T$ für die unbekannten Einträge v_{ij} von V_{ff}. Diese unterbestimmten Gleichungssysteme werden mit Hilfe des QR-Algorithmus gelöst, sodass man die normkleinste Lösung erhält. Die i.A. nicht hermitesche Matrix V_{ff} wird dann noch durch $\tilde{V}_{ff} = \frac{1}{2}(V_{ff} + V_{ff}^H)$ ersetzt. Damit erhält man eine neue Approximation $\tilde{\tilde{S}}$, die in gewissem Sinne am wenigsten von der alten Approximation $\tilde{S}$ abweicht, das Spektrum von S besser approximiert und weiterhin dünn besetzt, hermitesch und positiv definit ist.

Numerische Ergebnisse

Bei den Ergebnissen, die das Verfahren liefert sind vor allem die Schrittzahl des CG-Verfahrens mit der Ausgangsmatrix A_{ee} interessant, da diese Aufschluss über die Güte des verwendeten Präkonditionierers gibt. Desweiteren wird der Besetzung der Problemmatrizen in den verschiedenen Leveln Beachtung geschenkt, da sie ein Maß für den benötigten Rechenaufwand des jeweiligen Levels ist. Hier ist besonders interessant der Vergleich der Besetzungen mit Lumping und ohne Lumping. Relevante Größen und Einstellungen des Programms sind, wenn nicht explizit anders benannt oder gesetzt:
n: Gittergröße $\Rightarrow$ Problemmatrix $A_{ee} \in \mathbb{C}^{n^2/2 \times n^2/2}$
n_c: Anzahl der groben Gitterpunkte
nnz: Anzahl der Einträge verschieden von Null
$\delta = 10^{-3}$ Lumping-Parameter
$\beta = 0.2$ Steuerungsparameter für die Zufälligkeit im System
$\alpha_{CR} = 0.5$ Gewünschte asymptotische Konvergenzgeschwindigkeit der Relaxation im CR-Verfahren

Zunächst ein paar Daten zum veranschaulichen der Arbeitsweise des verwendeten CR-Algorithmus zur Bestimmung der Gitterzerlegung. $\alpha_{CR} = 0.5, 0.6$ gewünschte asymptotische Konvergenz der verwendeten Relaxation für Systeme der Bauart $A_{ff}x = b$

	$n_{CR_{0.5}}$	$\frac{n_{c_{0.5}}}{n}$	$n_{CR_{0.6}}$	$\frac{n_{c_{0.6}}}{n}$
$n = 64$	9	0.38	4	0.27
$n = 128$	11	0.35	5	0.24
$n = 192$	14	0.33	7	0.23



Die Bilder zeigen die Phasen eines Durchlaufes des CR-Algorithmus, man sieht sehr gut, wie der Algorithmus nach und nach die groben Gitterpunkte auswählt und zum bereits festgelegten groben Gitter hinzufügt.
Nach diesem kleinen Einblick in die Resultate des CR-Algorithmus nun zu den Ergebnissen des kompletten Verfahrens, also CG-Verfahren mit Multilevel Schur-Komplement Präkonditionierung. Verglichen werden die Ergebnisse mit und ohne Präkonditionierung, sowie mit einfacher Schur-Komplement Approximation ohne Probing (auch *triviale* Approximation genannt) und mit Probing.
Die ersten Tabellen zeigen zunächst Ergebnisse mit der Schur-Komplement Approximation ohne Probing

für verschiedene Problemgrößen mit $\alpha_{CR} = \frac{1}{2}$.

$n = 64$	$level0$	$level1$	$level2$	$level0$	$level1$	$level2$
n	2048	762	170	2048	762	158
nnz	18432	12816	4604	18432	12462	4604
$\frac{nnz}{n}$	9.0	16.82	27.08	9.0	16.35	8.03
n_{lump}	*ohne Lumping*			–	354	2592

$n = 192$						
n	18432	6145	1013	18432	6145	939
nnz	165888	95021	17381	165888	92645	4945
$\frac{nnz}{n}$	9.0	15.46	17.15	9.0	15.08	5.27
n_{lump}	*ohne Lumping*			–	2376	9908

Anschließend der Vergleich der Schur-Komplement Approximation ohne Probing und mit Probing, wobei $\alpha_{CR} = \frac{1}{2}$ ist.

$n = 128$	$level0$	$level1$	$level2$	$level0$	$level1$	$level2$
n	8192	2976	662	2192	2976	587
nnz	73728	49598	1472	73728	48322	3977
$\frac{nnz}{n}$	9.0	16.67	23.27	9.0	16.24	6.78
n_{lump}	*ohne Lumping*			–	1276	8634

$n = 128$	Probing App.					
n	8192	2976	433	8192	2976	371
nnz	73728	49598	7163	73728	44940	1725
$\frac{nnz}{n}$	9.0	16.67	16.54	9.0	15.1	4.65
n_{lump}	*ohne Lumping*			–	4658	3138

Abschließend der Vergleich der benötigten Schritte für das Level 0 CG-Verfahren mit $\alpha_{CR} = \frac{1}{2}$.

	ohne PK	PK 1-stufig		PK 2-stufig	
		$trivial$	$Probing$	$trivial$	$Probing$
$n = 64$	45	19	27	25	27
$n = 128$	31	18	28	26	28
$n = 192$	28	16	22	21	22

Es ist darauf hinzuweisen, dass insbesondere bei Anwendung der Probing-Korrektur die Konvergenz i.A. in einem 3-Level-Verfahren verloren geht und Gleiches auch bei Anwendung von Lumping bei beiden Approximation ab einem 4-Level-Verfahren zu beobachten ist. Wie man den Tabellen entnehmen kann, ist zu erwarten, dass die Besetzung der Matrix im Falle der Approximation ohne Probing und ohne Lumping immer weiter zunimmt, wenn man die Tiefe der Rekursion erhöht. Verwendet man Lumping, so ist zu befürchten, dass die Konvergenzeigenschaft verloren geht, auch wenn die Besetzungsdichte sehr gut aussieht. Die i.A. schlechteren Ergebnisse der Approximation mit Probing, sind vermutlich auf die doch sehr grobe Berechnung der Eigenvektoren zu den kleinen Eigenwerten zurückzuführen und spiegeln möglicherweise nicht die Möglichkeiten des Ansatzes wieder, der verwendet wurde.
Jedoch erkennt man, dass die Besetzung mit Probing auch ohne Lumping sich nicht so schlecht zu entwickeln scheint, wie im Fall ohne Probing. Sollte sich dieses Ergebnis auch mit den oben angesprochenen verbesserten Probing-Strategien reproduzieren lassen, so wäre der Ansatz vielversprechend.

Ausblick

Wie schon in den Ergebnissen erwähnt, wäre zunächst einmal zu schauen, ob man den Probing-Ansatz und die Anpassung an die Eigenvektoren zu kleinen Eigenwerten der Schur-Komplement Approximation verbessern kann. Sei es durch eine genauere Approximation der Vektoren, die Anpassung an mehrere Repräsentanten oder eine adaptive Anpassung, die solange neue Repräsentanten anpasst, bis die Kondition des präkonditionierten Systems $\tilde{S}^{-1}S$ gut genug ist.
Von großem Interesse ist natürlich auch die Frage ob sich das Verfahren auch auf 3- oder 4-dimensionale Probleme, wie sie in der QED bzw. QCD auftauchen, anwenden lässt und inwieweit dort der Lumping bzw. Probing Ansatz wirkt, da bei herkömmlichen Multilevelverfahren für diese Probleme Lumping oder etwas Vergleichbares immer zum Verschwinden der Konvergenz führt. Die damit verbundene Vergrößerung der Problemmatrix lässt dann auch bessere Aufschlüsse über das Verhalten des Verfahrens bei tieferer Rekursion zu und kann möglicherweise den oben beschriebenen Trend des übermäßigen Anwachsens der Besetzung ohne Probing und das sehr schöne Verhalten bei Einsatz des Probing-Ansatzes bestätigen.
Desweitern wäre es interessant zu testen, in wie weit man den Startaufwand, den man in die Berechnung der CF-Aufteilung steckt, auch für Systeme verwenden kann, die sehr ähnlich zum Ausgangssystem sind (z.B. geshiftet), um die sehr kostspielige Startphase besser auszunutzen.
Schlußendlich wäre, bei Bestätigung der positiven Eindrücke unter den angesprochenen Veränderungen noch eine Komplexitätsanalyse notwendig, um den Aufwand des Verfahrens besser beschreiben zu können und möglicherweise dann eine effiziente(re) Implementierung zu erzeugen.

Literatur

1. A.Frommer,
   A Schur complement preconditioner for random systems, unpublished
2. J.Brannick,
   Compatible relaxation and optimal interpolation, master theses unpublished

Ein kollaborativer Ansatz für die GUI-Programmierung

Björn Kuhlmann

RWTH Aachen

E-mail: bjoern.kuhlmann@rwth-aachen.de

Zusammenfassung: Es wurde ein Ansatz für die kollaborative Nutzung von GUI-Kontrollelementen entwickelt und erprobt. Der Ansatz wurde auf Basis einer vorhandenen Programmbibliothek für die Publikation von Events im Netzwerk (Event-Heap) sowie eines frei verfügbaren Toolkits für die GUI-Programmierung (FOX-Toolkit) entwickelt. Besonderes Augenmerk wurde auf die saubere Implementierung eines Listener-Threads auf Seite der zu vernetzenden Clients sowie auf die Entwicklung eines konsistenten Locking-Modells gelegt. Es wurde eine kleine Klassenbibliothek entwickelt, mit der vorhandene FOX-Applikationen ohne größeren Aufwand instrumentiert und zu einem kollaborativen Gesamtsystem vernetzt werden können.

Einleitung

Mit der rasant voranschreitenden Entwicklung der Netzwerktechnologie hin zu immer schnelleren und breitbandigeren Netzen werden in zunehmendem Maße neue Konzepte der netzbasierten Kollaboration interessant. Waren herkömmliche Internetanwendungen wie E-Mail, WWW oder auch Video-Streaming nicht oder nur bedingt von geringen Latenzzeiten abhängig, so rücken heute auch Anwendungen in den Blickpunkt, welche ohne echtzeitfähige Netzverbindungen nicht denkbar sind. So erfreuen sich beispielsweise Voice-over-IP und Online-Games immer größerer Beliebtheit.

Die dieser Arbeit zugrundeliegende Aufgabenstellung ist Teil des des KoDaVis-Projektes. Dieses Projekt beschäftigt sich mit der kollaborativen Visualisierung großer Datensätze aus der Atmosphärenchemie und ist eingebettet in dem nationalen Verbundprojekt VIOLA[2], einer gemeinsamen Anstrengung von Partnern aus Hochschulen, Forschungseinrichtungen, Industrieunternehmen und dem DFN-Verein. Ziel dieses Projektes ist es, in einem glasfaserbasierten Testbed neue Netztechniken und neue Formen der Netzintelligenz einzusetzen und sie, integriert mit entsprechend anspruchsvollen Anwendungen, zu erproben, um so den beteiligten Partnern Know-How für zukünftige Netzgenerationen zu vermitteln.

Das Ziel des KoDaVis-Projektes[1] ist die Entwicklung von Techniken für die netzbasierte kollaborative Visualisierung innerhalb räumlich verteilter Arbeitsgruppen und in einer heterogenen Infrastruktur. Hierbei unterscheiden sich die eingesetzten Rechner in Leistungsfähigkeit und Ausstattung. Die Visualisierung und Bearbeitung der Daten soll an bis zu vier verschiedenen Standorten gleichzeitig und gemeinsam durchgeführt werden. Für die Steuerung der Visualisierungssyteme, kommt u.a. eine graphische Benutzeroberfläche (GUI) zum Einsatz, die mit Hilfe des FOX-Toolkits erstellt worden ist. Aufgabenstellung dieser Arbeit war es, diverse Kontrollelemente einer solchen GUI zwischen mehreren Instanzen derselben Applikation zu synchronisieren. Die dabei gewonnenen Erkenntnisse sollen später bei der Entwicklung der KoDaVis-Clients genutzt und weiterentwickelt werden.

Eingesetzte Softwarekomponenten

Das FOX-Toolkit

Die C++-basierte Klassenbibliothek FOX [1] [4] ist ein Toolkit zur Entwicklung graphischer Benutzeroberflächen. Ursprünglich für Linux entwickelt, ist es heute für eine großen Anzahl verschiedener Plattformen verfügbar.

Es stellt eine wachsende Anzahl von Kontrollelementen, wie z.B. Schaltflächen, Textfelder, Grafikfelder und auch OpenGL-Kontrollelemente zur Manipulation von 3D-Graphiken, zur Verfügung. FOX implementiert weiterhin auch Icons, Bilder, Statuszeilenhilfe und Tooltips. Ein Hauptaugenmerk bei der Entwicklung von FOX ist Laufzeiteffizienz. Daher war es für uns wichtig, daß unsere kollaborativen Erweiterungen diese Eigenschaft nicht verletzen. Wir haben darauf geachtet, daß die Netzwerklatenz die Benutzbarkeit der GUI nicht beeinträchtigt.

FOX verwendet eine Anzahl von Techniken, um das Zeichnen und das räumliche GUI-Layout zu beschleunigen. Speicherbedarf wird minimiert, indem der Entwickler GUI-Elemente schnell erzeugen und zerstören kann. Obwohl FOX bereits eine große Sammlung von Kontrollelementen bereitstellt, bietet es dem Entwickler auch die Möglichkeit eigene zu erstellen. Das durchgängig objektorientierte Design ermöglicht es, eigene Klassen von bestehenden Klassen abzuleiten und so ausgewählte Features zu verändern bzw. hinzuzufügen.

Eines der Hauptziele beim Design des FOX-Toolkits war und ist, die Erstellung von Anwendercode so einfach wie möglich zu gestalten. So können Kontrollelemente oft mit nur einer Zeile C++-Code erzeugt werden. Sinnvolle Default-Werte vieler Konstruktor-Parameter erleichtern den Einstieg. Layoutmanager stellen sicher, daß sich die Entwickler nicht um die genaue Positionierung der GUI-Elemente sorgen müssen. Eine weitere positive Eigenschaft von FOX, welche die Anzahl zu schreibender Codezeilen verringert, ist das Message/Target-basierte Design. Alle GUI-Events werden als Nachrichten zwischen Objekten ausgetauscht. Durch diese Art Verknüpfen von Kontrollelementen untereinander und mit dem Rest der Applikation kann viel Quellcode vermieden werden. (Wir erinnern uns: Der beste Code ist der, den man *nicht* schreibt!) So kann ein FOX-Kontrollelement direkt als Target von einem anderen angegeben werden. Solche Verbindungen können auch bidirektional sein. Weiterhin ist es in FOX besonders einfach, den Zustand der GUI aktuell zu halten. Die einzelnen Kontrollelemente können sich automatisch aktualisieren indem sie den aktuellen Zustand der Anwendung abfragen. So können sich z.B. einzelne Kontrollelemente automatisch ausgrauen, falls sie zur Zeit nicht verfügbar sind.

Event-Heap

Der Event-Heap [3] ist eine TCP/IP-basierte Middleware zur Koordinierung von Ereignissen in verschiedenen Anwendungen. Dabei werden Events über das Netzwerk verschickt. Der Event-Heap basiert auf einer Client-Server-Architektur. Ein Server-Prozeß koordiniert den Austausch von frei definierbaren Ereignissen zwischen den Clients. Die Clients melden sich beim Event-Heap an, und können ihre Events veröffentlichen oder Events von anderen Clients empfangen. Die Event-Heap-API ist für diversen Sprachen (u.a. C++) und Plattformen verfügbar.

Abbildung 1 zeigt die Struktur einer Event-Heap basierten Vernetzung. Nachdem der Event-Heap auf einem beliebigen Rechner als Serverdienst gestartet wurde, verbinden sich die zu synchronisierenden Anwendungen (Application 1-3) mit diesem.

Einzelne Anwendungen können nun beim Event-Heap Events eines bestimmten Typs „abonnieren“, d.h. sich auf die Liste der Interessenten für diese Events setzen lassen. Im dargestellten Beispiel abon-

[1] **F**ree **O**bjects for **X**

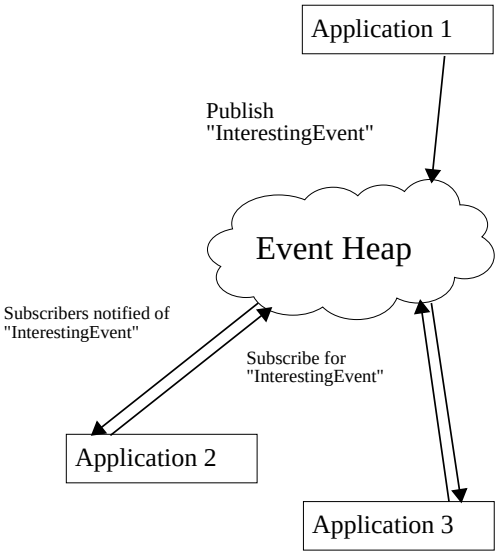


Abbildung 1: Beispiel Event-Heap

nieren die Anwendungen 2 und 3 Events vom Typ „InterestingEvent“.

Liegt nun in einer der Anwendungen ein Event vor, welches veröffentlicht werden soll, wird dieses Event an den Event-Heap geschickt. Dieser stellt fest, von welchem Typ das hereinkommende Event ist und leitet es an alle Anwendungen weiter, die diesen Event-Typ abonniert haben. Im Beispiel sendet Anwendung 1 ein Event vom Typ „InterestingEvent“an den Event-Heap und dieser verschickt es an die Anwendungen 2 und 3.

POSIX-Threads

Zur Anbindung einer FOX-GUI an den Event-Heap ist es notwendig, zwei Aufgaben gleichzeitig abzuarbeiten. Ein Thread bedient die GUI, der andere empfängt Nachrichten vom Event-Heap und verarbeitet diese. Hierfür kamen Pthreads zum Einsatz. Pthreads sind für die Plattformen, auf denen die spätere Anwendung laufen wird, verfügbar. Der Pthread Standard wurde 1995 in ANSI/IEEE POSIX 1003.1 festgelegt.

Da das FOX-Toolkit nicht thread-safe ist, wurde das Verfahren des gegenseitigen Ausschluß (mutual exclusion, kurz: mutex) verwendet. Damit wird verhindert, daß nebenläufige Prozesse gleichzeitig auf Daten zugreifen und so unter Umständen inkonsistente Zustände herbeiführen. Mutex-Variablen dienen der Serialisierung des Zugangs verschiedener Pthreads zu kritschen Bereichen. Sie stehen im Allgemeinem im Namensraum mehrerer Threadfunktionen und können über spezielle Funktionen gelockt bzw. geunlockt werden. Lock beginnt einen atomaren Bereich und unlock beendet ihn. Falls ein Thread sich in einem atomaren Bereich befindet (zwischen lock und unlock) wird ein anderer Thread, der einen lock verlangt, blockiert. Nach Ausführung von unlock erhält genau ein anderer Thread, der auf einen lock wartet, den lock. Welcher das ist, wird vom Betriebssystem festgelegt. Durch die Verwendung von Mutexen wurde sichergestellt, daß nur ein Thread die GUI manipulieren kann. Somit wird verhindert, daß es zu inkonsistenten Zuständen der gesamten Applikation kommt.

Anbindung einer FOX-GUI an den Event-Heap

Instrumentierung der Applikation

Zur Anbindung einer FOX-Applikation an den Event-Heap muss der bestehende Code um einige wenige Komponenten erweitert werden. Zunächst wird die `MainWindow`-Klasse mittels Vererbung um einige zusätzliche Definitionen erweitert.

Als nächstes wird ein Kontrollelement instanziert, welches die Adresse des Event-Heap-Servers (EHS) einliest und Schaltflächen für den Auf- und Abbau der Verbindung bereitstellt. Ein erfolgreicher Verbindungsaufbau initiiert dabei einen POSIX-Thread, welcher für das Entgegennehmen der Events vom EHS notwendig ist.

Abbildung 2: Event-Heap-Connection-Widget

Für die Serialisierung der Events bzw. der ihnen zugeordneten Parameter muss der Anwender eigene Callback-Funktionen implementieren. Die Deserialisierung erfolgt in einer weiteren, vom Anwender bereitzustellenden, Funktion.

Abschließend sind alle Einträge in den FOX-Messagemaps, welche sich auf kollaborativ zu nutzende GUI-Funktionen beziehen, abzuändern. Die dort deklarierten Callback-Funktionen werden durch die eigenen Event-Serialisierungen ersetzt. Damit ist die Instrumentierung des Codes abgeschlossen. Mehrere Instanzen der Applikation können über den Event-Heap synchronisiert werden.

Beispiel zur Instrumentierung

Die nötige Instrumentierung soll nun anhand eines kleinen Beispiels näher erläutert werden. Hierzu wird im folgenden gezeigt, wie eine Schaltfläche kollaborativ zu Verfügung gestellt werden kann.

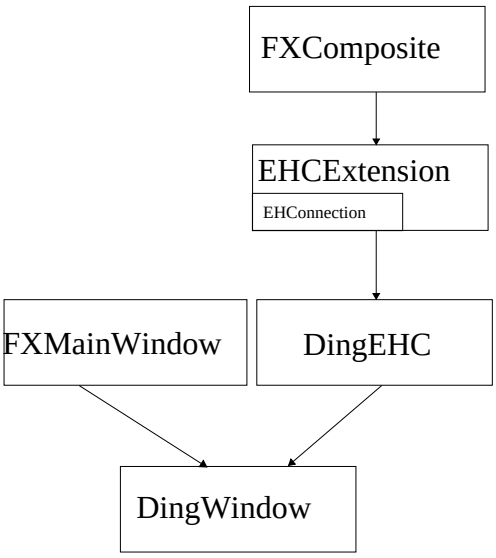


Abbildung 3: Klassendiagramm

Zunächst wird von der Klasse `EHCExtension` eine Ableitung erstellt, in welcher die zu verwendenden Serialisierungfunktionen sowie die Deserialisierungsfunktion bekanntgemacht werden. Die Basisklasse `EHCExtension` stellt unter anderem einen Zeiger auf das lokale Event-Heap-Objekt zur Verfügung.

```
class DingEHC : public EHCExtension {

 public:
    long onCmdButtonDing_eh(FXObject *, FXSelector, void *);

    [...]

 protected:
    static void deserialize(void *tgt, int id);
};
```

Als nächstes ist die vom Anwender sowieso zu erstellende Ableitung der `FXMainWindow`-Klasse um die Event-Heap Schnittstellen zu erweitern:

```
class DingWindow :
  public FXMainWindow,
  public DingEHC // <=== !!!
{
   [...]
}
```

In der Message-Map werden alle Callback-Funktionen durch die vom Anwender bereitzustellenden Serialisierungsfunktionen ersetzt:

```
FXDEFMAP(DingWindow) DingWindowMap[] = {
   FXMAPFUNC(SEL_COMMAND,
             DingWindow::ID_DING,
          /* DingWindow::onCmdButtonDing), */
             DingWindow::onCmdButtonDing_eh),
};
```

Nun muss im Konstruktor der GUI noch das eigentliche Connector-Kontrollelement erzeugt werden. Diesem wird ein Zeiger auf die bei einlaufenden Events aufzurufende Deserialisierungsfunktion sowie auf das Ziel der FOX-Messages übergeben:

```
eh_connector = new EHConnector(frame, DingEHC::deserialize, this);
```

Damit sind alle für die kollaborative Nutzung der GUI-Kontrollelemente notwendigen Änderungen am vorhandenen Code einer FOX-Applikation vorgenommen.

Für die Serialisierung der Events bzw. der mit diesen verbundenen Parameter muß der Entwickler die neue CallBack-Routine `onCmdButtonDing_eh()` schreiben. Im Falle eines einfachen Klicks auf eine Schaltfläche wird hier lediglich die entsprechende Message-ID ohne weitere Parameter an den Event-Heap geschickt:

```
long DingEHC::onCmdButtonDing_eh(FXObject *obj,
                                 FXSelector selector,
                                 void *tgt)
{
    EHButtonClick(DingWindow::ID_DING);
    return 1;
}
```

Die hier verwendete Message-ID steht in allen Instanzen der Applikation mit dem selben numerischen Wert zur Verfügung und kann daher auch zur Bezeichnung von Events in der vernetzten Applikation genutzt werden.

Die vom Event-Heap empfangenen Events müssen nun noch die lokalen Callback-Funktionen aufrufen. Dazu müssen evtl. zusätzliche Parameter deserialisiert werden. Für diese Aufgabe hat der Anwender eine entsprechende Deserialisierungsfunktion zur Verfügung zu stellen. In unserem Beispiel würde diese wie folgt aussehen:

```
void DingEHC::deserialize(void *tgt, int id)
{
    DingWindow *target = (DingWindow*)tgt;

    EHMutexLock();

    switch(id)
    {
        case DingWindow::ID_DING:
            target->onCmdButtonDing(NULL, 0, NULL);
            break;

        default:
            break;
    }

    EHMutexUnlock();
}
```

Da diese Funktion aus dem Thread heraus aufgerufen wird, welcher die Nachrichten vom Event-Heap entgegennimmt, müssen hier vorgenommene GUI-Manipulationen (in `onCmdButtonDing()`) mittels Mutex-Locks vom eigentlichen GUI-Thread separiert werden.

Funktionsweise der Instrumentierten Applikation

Alle Anfragen, auch die eigenen, werden zunächst zum Event-Heap geschickt und dann vom `deserialize()`-Thread empfangen. Dadurch wird sichergestellt, daß sich alle GUIs immer im gleichen Zustand befinden. Tests zeigten, daß dies zu keiner relevanten Verzögerung führte. Dabei lief der Event-Heap auf einem Rechner im VIOLA-Netz in Aachen und die beiden GUIs auf verschiedenen Rechnern im FZ Jülich.

Anwendbarkeitstest

Um die Verwendung des Event-Heaps in Verbindung mit dem FOX-Toolkit zu testen, entstand eine Demo-Applikation (s. Abb. 4).

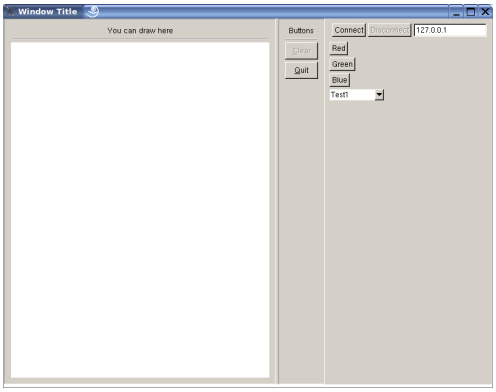


Abbildung 4: Demo-Applikation

In dieser Demo-Applikation wurde getestet, welche Kontrollelemente sich zur Synchronisation über den Event-Heap eignen. Hierzu zählten u.a. Schaltflächen, Comboboxes und Zeichenfelder. Zunächst wurden nur einige Schaltflächen über eine gemeinsame Verbindung zum Senden und Empfangen verschickt. Dabei kam es jedoch teilweise zu Störungen, d.h. das Programm brach mit undefinierten Empfangsereignissen ab.

Eine zweite Verbindung, also eine Verbindung für den GUI-Thread zum Senden von Events und eine Verbindung für den Deserialize-Thread zum Empfangen von Events, schaffte Abhilfe.

Bei der gemeinsamen Verwendung der Verbindung für Sende- und Empfangthread kam es zu ungültigen Sende- und Empfangsereignissen. Diese entstanden, da ein gemeinsamer Puffer verwendet wurde, und während ein Thread empfing, ein anderer Thread sendete und somit Daten in den gemeinsamen Puffer schrieb. Durch eine weitere Verbindung werden zwei verschiedene Puffer verwendet und es können keine Überschreibungen und somit keine ungültigen Ereignisse entstehen.

Nachdem das Versenden von Schaltflächen-Events funktionierte, wurde die Demo-Applikation um eine Combo-Box erweitert. Bei ersten Tests mit einer naiven Implementierung kam es erwartungsgemäß zu inkonsistenten Zuständen wenn zwei Benutzer zeitgleich verschiedene Elemente aus der Combo-Box auswählten. An diesem Punkt der Entwicklung wurde es also notwendig, ein intelligentes Locking-Schema für den Austausch von Nachrichten zwischen den Applikationen und dem Event-Heap zu entwerfen. Dieser, im Folgenden besprochene, Lock-Mechanismus ist dabei strikt von dem weiter oben beschriebenen Pthread-Lock-Mechanismus zu unterscheiden, welcher nur den Zustand einer einzelnen GUI konsistent hält.

Zur Lösung des Problems waren i.W. zwei Dinge zu tun. Zunächst einmal muss jeder lokale Callback-Aufruf über den Event-Heap-Server laufen. Wird also beispielsweise auf einem Client, der mit dem EHS verbunden ist, eine Checkbox angeklickt, so wird die entsprechende Update-Funktion auf diesem wie auf allen anderen Clients, von dem vom EHS replizierten Event getriggert. Auf diese Weise ist sichergestellt, daß es nur einen Punkt im Kontrollfluß eines jeden Clients gibt, an dem kritische Callback-Aufrufe auf ihre Gültigkeit im Kontext der kollaborativen Anwendung geprüft werden können.

Um zu verstehen, was ein gültiges und was ein ungültiges Event in diesem Zusammenhang ist, betrachten wir zwei User, Anton und Berta, mit je einer Instanz eines Clients mit einer einfachen Checkbox. Beide

Clients sind über den Event-Heap-Server miteinander verbunden und die Checkbox ist auf beiden GUI's „unchecked". Angenommen Anton und Berta klicken gleichzeitig auf ihre Checkbox. Dann sind sie sich offensichtlich darüber einig, daß ihre gemeinsame Checkbox fortan „checked"sein soll. Tatsächlich erhält aber jede der beiden GUI's *zwei* toggle-Signale! Würden beide einen Callback-Aufruf triggern, so wäre im Ergebnis die Checkbox wieder „unchecked". In unserem Ansatz soll nun genau eines der beiden Events als ungültig markiert werden.

Das von uns entworfene Schema arbeitet wie folgt. Ein Event ist genau dann gültig, wenn erstens vom Absender des Events zuvor ein (gültiges) Lock-Event empfangen wurde und zweitens seitdem noch *kein* (gültiges) Unlock-Event empfangen wurde. Kurz, ein Event ist gültig, wenn der Absender das globale Lock besitzt. Die eindeutige Reihenfolge, mit der die Events vom Server repliziert werden zusammen mit der stromorientierten Verbindung vom Server zu den Clients garantiert, daß es immer nur einen Besitzer des Locks gibt.

Die Latenz beim Aufruf lokaler Callback-Funktionen, welche durch den „Umweg"über den EHS entsteht, war in unseren Tests nicht wahrnehmbar. Wir haben diese Tests mit bis zu 2 Clients und Verbindungen über das VIOLA-Netz (vom FZ-Jülich nach RWTH-Aachen und zurück) durchgeführt.

Der oben beschriebene Lock-Mechanismus wurden in leicht abgewandelter Form auch für die Übertragung von Mouse-Events verwendet. Um die Übertragung von Mouse-Events zu testen, haben wir die Demo-Applikation um ein Zeichenfeld erweitert. In diesem hat der Benutzer die Möglichkeit eine Freihandzeichnung zu machen. Damit immer nur ein Anwender zeichnen kann, wurde beim `mouseDown`-Event das globale Lock gesetzt, und beim `mouseUp`-Event wieder freigegeben. Die weiteren `mouseMove`-Events wurden zwischen dem `mouseDown`- und dem `mouseUp`-Event verschickt. Dadurch konnte sichergestellt werden, daß der Anwender, welcher die GUI zur Zeit bedient, nicht durch andere Anwender unterbrochen werden kann. Bei der Übertragung dieser `mouseMove`-Events mußten wir allerdings feststellen, daß es zu einer Verzögerung zwischen der Mausbewegung und der Darstellung der so gezeichneten Linie kam. Wir haben nicht weiter untersucht ob diese Verzögerung im EHS bei der Replikation der Events entsteht oder ob es auf Seite der Clients beim Entgegennehmen und weiterverarbeiten der Events zu einem Stau kommt. Vielmehr haben wir für den Fall das Events die in schneller Folge auftreten einen Downsize-Faktor eingeführt, der es ermöglicht aus einer Serie von gleichartigen Events nur noch jedes n-te Event über den Server zu publizieren.

Im Ergebnis wurden die auf diese Weise auf das Zeichenfeld gemalten Linien ein wenig „eckiger", aber dafür wurden sie wieder „in Echtzeit"dargestellt. Dieser Faktor hilft sowohl wenn der Server zu langsam ist solche Event-Folgen schnell genug zu replizieren, als auch wenn die Clients zu langsam sind die Events in der notwendigen Geschwindigkeit entgegenzunehmen. Wir haben mit einem Faktor n=5 hinreichend glatte Linien ohne merkliche Verzögerung über den EHS zeichnen können.

Zu untersuchen bleibt hier, ob es sinnvoll ist 2 Downsize-Faktoren, nämlich einen für die Folge der herausgehenden Events und eine weiteren für die Folge der hereinkommenden Events zu definieren. Mit ersterem könnte ein Client bei Bedarf den EHS „schonen", mit dem zweiten sich selbst vor zu schnell hereinkommenden Events schützen.

Zusammenfassung und Ausblick

Es wurde in dieser Arbeit ein Kollaborationsmodell getestet. Es hat sich gezeigt, daß ein einfacher Event-Replikator ausreicht um beliebige Events zwischen mehreren Clients zu synchronisieren. Ein konsistentes Locking-Schema kann vollständig auf Seite der Clients implementiert werden.

Es hat sich weiterhin gezeigt, daß das Softwarepaket *Event Heap* voll einsatzfähig ist und im Rahmen des KoDaVis-Projektes verwendet werden kann.

Bei der Implementierung der Test-Applikation hat sich gezeigt, daß das FOX-Toolkit eine schnelle und unkomplizierte Anbindung an einen Event-Replikator ermöglicht. Wir konnten einen Ansatz für eine Klassenbibliothek entwickeln, mit der vorhandene Applikationen mit minimalem Aufwand für den Einsatz in einer kollaborativen Umgebung instrumentiert werden können.

Die im Rahmen dieser Arbeit gewonnenen Erkenntnisse können nun bei der Weiterentwicklung des KoDaVis-Clients verwendet werden.

Zur Zeit hängt die Kommunikationsschnittstelle von der Event-Heap-Architektur und vom FOX-Toolkit ab. Eine höhere Abstraktion wäre nötig, um diese Abhängigkeiten aufzulösen und es so einfach zu ermöglichen, zwischen verschiedenen Netzwerkschnittstellen einerseits und diversen GUI-Toolkits andererseits auswählen zu können.

Danksagung

Bedanken möchte ich mich bei allen, die mich bei meiner Arbeit hier am Forschungszentrum Jülich tatkräftig unterstützt haben. Mein ganz besonderer Dank geht an Thomas Düssel, der mich betreut hat, an Ingo Assenmacher, an Prof. PhD Christian Bischof, der sich sehr für mich eingesetzt hat, an Rüdiger Esser und an das NIC.

Literatur

1. Kodavis.
   http://www.viola-testbed.de/content/index.php?id=kodavis0.
2. Viola.
   http://www.viola-testbed.de.
3. Rafael Ballagas, Meredith Ringel, Maureen Stone, and Jan Borchers. istuff: a physical user interface toolkit for ubiquitous computing environments. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 537–544, New York, NY, USA, 2003. ACM Press.
4. Jeroen van der Zijp. Fox-toolkit.
   http://www.fox-toolkit.org.

Performance of the ScaLAPACK Eigensolver PDSYEVX on IBM Blue Gene/L
– First Results –

Robert Speck

University of Trier
Department of Mathematics

E-mail: robertspeck@gmx.net

Abstract:

The 2048 processor IBM Blue Gene/L system at the John von Neumann Institute, Research Center Juelich, is the first installation of this innovative and powerful computer architecture in Germany. It aims at the investigation of highly scalable simulation problems like quantum chromodynamics (QCD) and other high-performance applications. In this project the performance of Blue Gene /L regarding the important and fundamental ScaLAPACK routine `PDSYEVX` is analyzed and discussed. Furthermore comparisons of the p690 IBM eServer Cluster 1600 and Blue Gene/L are drawn and complications are examined.

IBM Blue Gene/L

This section gives a short introduction to the Juelich IBM Blue Gene/L system ("JUBL" / "BG/L") architecture and software. It gives an overview to the highlights and specialties of the machine and the different packages used in this project.

Configuration

Blue Gene/L's base component (the compute node) is a 32-bit IBM PowerPC 440 dual-core chip with 700MHz and a double 64-bit Float-Point Unit (FPU). Each FPU can execute up to two multiply-adds (FMA) per cycle, meaning that the peak performance is eight 64-bit floating-point operations per cycle, resulting in 2.8 GFLOPS per core (i.e. processor) and 5.6 GFLOPS per node.
Each chip can dispose of 512 MByte RAM, 32 KB instruction and 32 KB data first-level (L1) cache. Moreover there are two independent 2 KB second-level (L2) caches, one 16 KB multiported Scratch SRAM buffer and 4 MB of shared embedded EDRAM as third-level (L3) cache. 32 chips with associated memory are plugged on a compute node book, whereas 16 node books form a midplane and a rack is made of 2 midplanes with 512 GByte RAM [1]. According to this scheme each rack possesses 2048 processors so that the theoretical peak performance is 5.6 TFLOPS per rack (LINPACK measurement provides a performance of 4.7 TFLOPS) [2].
As a compromise of efficiency and usability each node can use 512 MByte RAM, but no virtual memory, and can operate in two different modes. Using the coprocessor-mode computation and communication are separated. The computing processor can access the whole memory and bandwidth, whereas the other processor attends to communication. Using the virtual-node-mode each processor acts as an independent node, which gives every processor the ability to communicate and to compute. Here the memory and

bandwidth has to be shared so that the amount of memory per computing processor is limited to 256 MByte RAM. The theoretical peak performance can only be reached if both processors compute, i.e. if the whole rack uses the VN-mode.
In addition to the compute nodes the system consists of 64 I/O-nodes (two per node book) with an external 1-GB/sec-Ethernet connection, a service node used to control the system and a front-end node, which manages user logins, job submits and cross-compiling. Among the external 2.2 TByte filesystem BG/L-users can access their HOME- and WORK-directory provided by the other supercomputer p690 IBM eServer Cluster 1600 ("JUMP") in Juelich.
There are five different interconnection networks, e.g. a 3D torus or a global tree. The 3-dimensional torus can only be used with 512 nodes, below this amount the standard topology is a 2D mesh. On JUBL one of the midplanes has to be mounted as a whole (in which the 3D torus is the standard setup), the other one can be divided into 16 compute books or 4 partitions containing 4 compute books. Using a different amount of nodes will result in allocated but redundant nodes.

Software Versions

The system software consists of two different kernels: The compute node kernel and the I/O-node kernel. The kernel of the compute node provides a small, simple Linux-like run time environment. This kernel is a single user, single process run time system and has no paging mechanism. The kernel of the I/O-node is a Linux port with specific patches for the Blue Gene architecture [1].
Since September 2005 the first official release of BG/L software (V1R1M0) which includes e.g. compiler- and communication-routine-upgrades is running [3]. The following compilation gives a detailed view of the software and their versions (including potential changes due to the installation of the official release) used for this project:

- xlf and xlf90: The IBM XL Fortran Compilers offer full support for the Fortran 77 and 90 language standards [4]. In addition to these standard ppc4 compilers there is a JUBL-specific runtime add-on. Their versions changed with the upgrade from 9.1.0-02 to 9.1.0-04.
- MPI: The Message Passing Interface is a widely used standard for writing message-passing programs. It is designed to give users control of massively parallel machines and workstation clusters [5]. The MPI used by JUBL is a MPICH2 variant, version bglmpi-05.202.1-2.ppc64 (changed after the upgrade to bglmpi-2005.1.1-0.ppc64).
- BLAS and dgemm.rts: The Basic Linear Algebra Subprograms perform high quality vector and matrix operations [6]. The JUBL-adapted dgemm routine (rts = run time system) is used for computation of the matrix-matrix-multiplication $C = \alpha AB + \beta C$, where A, B and C are matrices and α, β are scalars [7]. On JUBL these routines are unsupported versions of BLAS440 and DGEMM which were not modified by the upgrade [8], but they will be included in the first release of ESSL.
- BLACS: The Basic Linear Algebra Communication Subprograms contain routines to make linear algebra applications easier to program and more portable. This project's goal is to create a linear algebra oriented message passing interface that can be implemented efficiently and uniformly across a large range of distributed memory platforms. The BLACS are used as the communication layer for ScaLAPACK [9]. On JUBL the public domain version 1.1 is installed [8], but these routines rely on MPI, i.e. if the program is linked after the upgrade, the new version of MPI is used.
- ScaLAPACK: The Scalable Linear Algebra Package is a library of high-performance linear algebra routines based on LAPACK and BLAS. It contains parallel routines for solving systems of linear equations, least squares problems and eigenvalue problems [10]. JUBL-users can work with public domain version 1.7 [8], but depending on BLACS this package uses the new MPI-routines after the upgrade.

Most of the results in this work have been produced before the upgrade, but tests with the official release did not spot essential differences. Especially recompiling ScaLAPACK with the new compiler add-ons has shown no effect.

Programming Details

The aim of this section is to describe the mathematical and technical background of the PDSYEVX routine in the way it has been used in this project and to explain the testing program with its specific characteristics.

The routine `PDSYEVX`

The ScaLAPACK routine `PDSYEVX` plays a decisive role in quantum chemistry applications or for solving optimization- or boundary-value-problems. The routine computes selected eigenvalues and optionally eigenvectors of a real symmetric matrix A. Eigenvalues and eigenvectors can be selected by specifying a range of values or a range of indices. This routine is an expert driver which calls the necessary ScaLAPACK subroutines in three computation steps:

i. Reduction of A to tridiagonal form using Householder transformation (call `PDSYNTRD`)

ii. Computation of the eigenvalues using bisection (call `PDSTEBZ`)

iii. Computation of the eigenvectors using inverse iteration and back-transformation (call `PDSTEIN` and `PDORMTR`)

According to its relevance the routine `PDSTEIN` has to be described separately. After some initial tests the algorithm calls the sequential routine `DSTEIN2` which is a variant of the LAPACK `DSTEIN` and computes the eigenvectors of A corresponding to the specified eigenvalues via inverse iteration. The maximum number of iteration steps is limited by an internal parameter which is set to 5. `DSTEIN2` has to be invoked for each eigenvector. Finally the routine performs a simple redistribution as well as sorting and permutation operations.
Like the corresponding LAPACK routine `DSYEVX` `PDSYEVX` tries to reorthogonalize eigenvectors belonging to clustered eigenvalues. In contrary to `DSTEIN` this is done by `DSTEIN2`, too. With an additional variable the user can specify which eigenvectors should be reorthogonalized. The reorthogonalization of all eigenvectors belonging to one cluster of eigenvalues has to be done on a single node, which is both an serial bottleneck and a restriction of the available workspace [11]. To avoid accordant problems the matrix size is limited to 5000 in this project.

The testing program

The testing program written in FORTRAN90 starts with an initialization of specific values, which will not change during the measurements. After the initialization of MPI and BLACS the grid-shape $NPROW \times NPCOL$ (the number of processors P equals $NPROW \cdot NPCOL$), the range for the eigenvalues $[RLB, RUB]$ and two values for measurement purpose are set. Moreover constants for starting value of the matrix dimension, increment and an iteration factor are defined.
With these values the program enters a loop which increases the matrix dimension with each cycle. This structure includes four sections:

i. Construction: First of all the matrix size N and the block size NB are defined, where N depends on the iteration step and the starting vales. The $N \times N$ matrix is divided into $NB \times NB$

blocks which are mapped to the processor grid (see Fig. 1). After generating N eigenvalues inside $[RLB, RUB]$ and a Householder reflector Q with `RANDOM_NUMBER` the dense matrix A is generated via $A = Q^H DQ$ where D is a matrix whose entries d_{ii} equals one of the chosen eigenvalues and $d_{ij} = 0$ for $i, j = 1, \ldots, N$, $i \neq j$. The matrix is distributed with the two-dimensional block-cyclic distribution scheme (shown in Fig. 1) which is the data layout that is widely used in the ScaLAPACK library for dense matrix computation.

ii. Memory tests: The expert driver `PDSYEVX` can run in a special, non-computing mode where the routine compares the amount of needed and given memory. This test is done twice: First the routine checks for adequate memory for finding the eigenvalues of A, second `PDSYEVX` simulates the reorthogonalization of the corresponding eigenvectors. Afterwards sufficient workspace is allocated.

iii. Computation: Here `PDSYEVX` computes all eigenvalues and eigenvectors. The execution time is measured by calling `rts_get_timebase` which returns the exact number of cycles each processor has made. This value divided by the clockspeed of 700MHz gives the exact time used for computation.

iv. Analysis: In this final step different tests verify the results. The program also checks for the accuracy of the computed eigenvalues by comparing them with the previously generated ones.

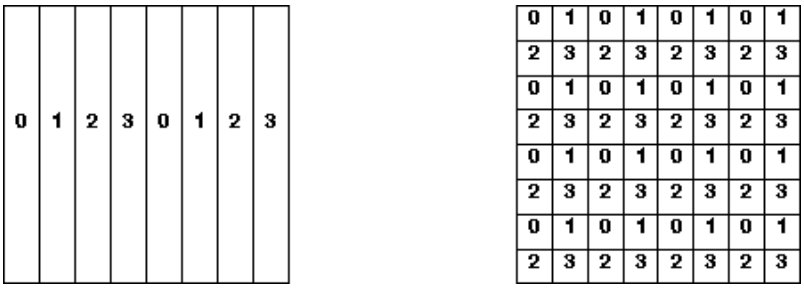


Figure 1: The one-dimensional block-cyclic column-distribution with $NPROW = 1,\ NPCOL = 4$ and the two-dimensional block-cyclic distribution with $NPROW = NPCOL = 2$ [12]

It is important to mention that each loop cycle generates its own matrix and eigenvalues by invoking the routine `RANDOM_NUMBER`, because the results of this routine depend on how often it has been started before. After the loop is finished BLACS and MPI are terminated.

Compiling and Running

To compile the testing program different calls of the xlf/xlf90 compilers are necessary. The Makefile compiles the program with the flags

```
-O0 -g -I$(BGLSYS)/include -L$(BGLSYS)/lib -qarch=440 -qtune=440
```

where `-O0` is the lowest degree of optimization, and links it with the precompiled ScaLAPACK, BLACS, BLAS and MPI routines. The ScaLAPACK library has been compiled with `-O0`, too. For testing purpose the routines `PDSYEVX`, `PDSTEIN`, `DSTEIN2` and the vector-product-routine `DDOT` are compiled separately, e.g. for the instrumentation workaround (see "JUMP vs. JUBL") or for testing optimization flags (which has not shown a relevant progress).

The command to run the executable program is a `mpirun` call with different flags and options:

```
mpirun -partition X -mode Y -np Z -exe ‘/bin/pwd‘/scaleigtestx.rts
                          -cwd ‘/bin/pwd‘
```

Here `X` means a specific partition, `Y` switches between the different modes `VN / CO` and `Z` specifies a number of processors. The flag `-mode` is optional and it is possible to set just one of the `-np` or `-partition` flags. `scaleigtestx.rts` is the executable which is sent to the compute nodes by mpirun.

Performance Analysis

Model Initialization

Before starting the main part of measurement it is important to find a consistent setup and parameters, that influence the results. This part discusses different grid-shapes, increasing block sizes and the two node-modes.

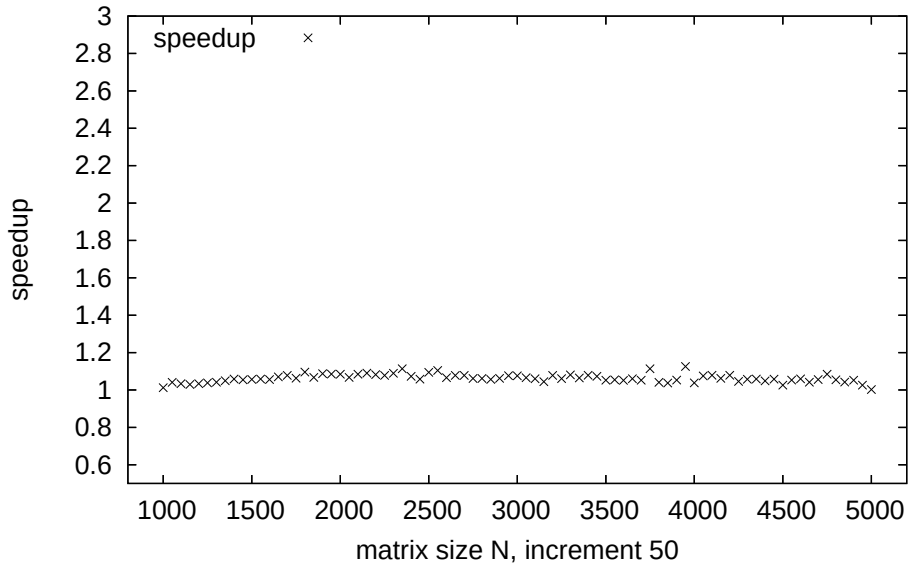


Figure 2: Comparison of VN- and CO-mode: $P = 64,\ 8 \times 8,\ N = 1000, \ldots, 5000,\ NB = 20$

In Fig. 2 the coprocessor- is compared with the virtual-node-mode using 64 processors which are arranged on an 8×8 grid. The matrix size varies from $N = 1000$ to $N = 5000$ with a block size of 20. To compute the speedup, the time needed in VN-mode is divided by the time needed in CO-mode . The computed values of the speedup curve are located next to 1.1 which means that the CO-mode is a little bit faster than the VN-mode. Normally the VN-mode is faster than the CO-mode for a given number of nodes and it makes better use of the machine, because both processors are used for computation, but the memory per node is relatively small. For unity reason the subsequent results base on the CO-mode. Every measurement which uses the VN-mode, e.g. the long-run analysis, will be indicated with a (VN). Fig. 3 shows three different values for the block size NB using 32 processors and matrix sizes of $N = 1000, \ldots, 4000$. It is obvious that there are no significant differences between the three values of NB. Even the noticeable structure starting at $N = 3700$ does not change with modified block size.

This structure will appear in most of the following measurements again. As a convention the block size $NB = 20$ is chosen so that there will be values of N which are not divisible by NB without remainder (e.g. $N = 2050$) to check their effect on the results.

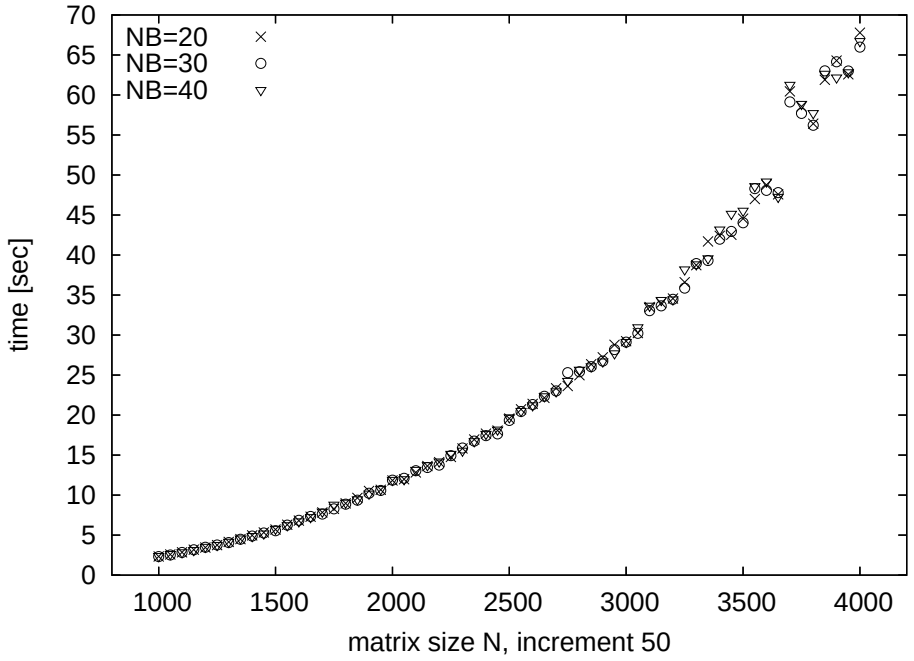


Figure 3: Comparison of block sizes: $P = 32,\ 8 \times 4,\ N = 1000, \ldots, 4000$

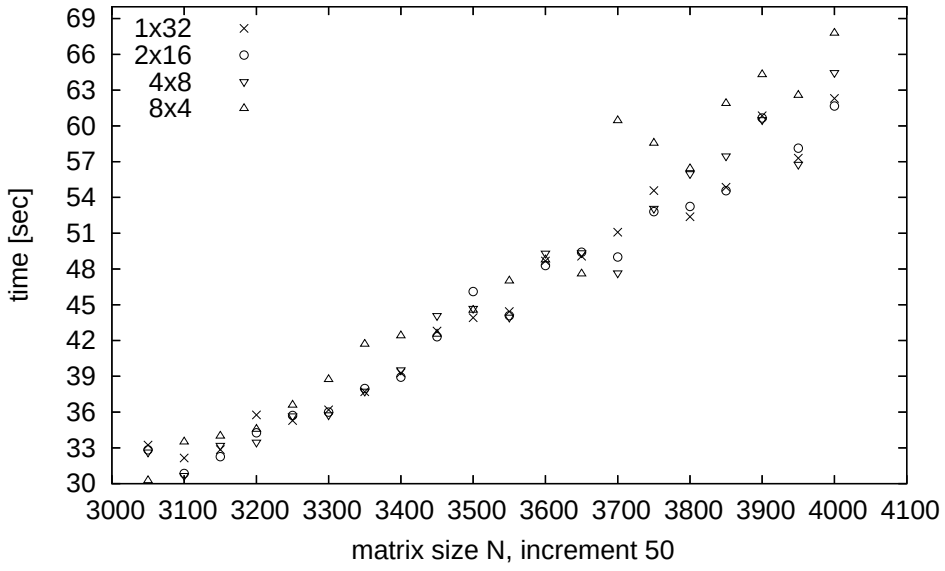


Figure 4: Comparison of grid-shapes: $P = 32,\ N = 1000, \ldots, 4000,\ NB = 20$ (extract)

The comparison of different grid-shapes reveals an unexpected result:
Fig. 4 shows an extract of a test series using 32 processors, matrix sizes of $N = 1000$ up to $N = 4000$ and a block size of 20. Naive and well balanced values for $NPROW$ and $NPCOL$ produce the struc-

ture mentioned before again whereas an unbalanced grid-shape harmonizes these value. Although well balanced grids produce these outliers the subsequent results base on native grid-shapes (e.g. $64 = 8 \times 8$, $128 = 16 \times 8$) to compare and examine the occurrence of this structure.

Long-run analysis

The following tests try to examine which values are unfavorable for the speedup analysis, i.e. for usability purpose. Different values for the matrix size N and the block size NB are checked with 64 processors using the VN-mode. N varies from 2000 to 3080 with an increment of 1. Fig. 5 represents $NB = 20$, Fig. 6 represents $NB = 21$. The plots are divided into odd and even values of N to check their influence on the execution times.

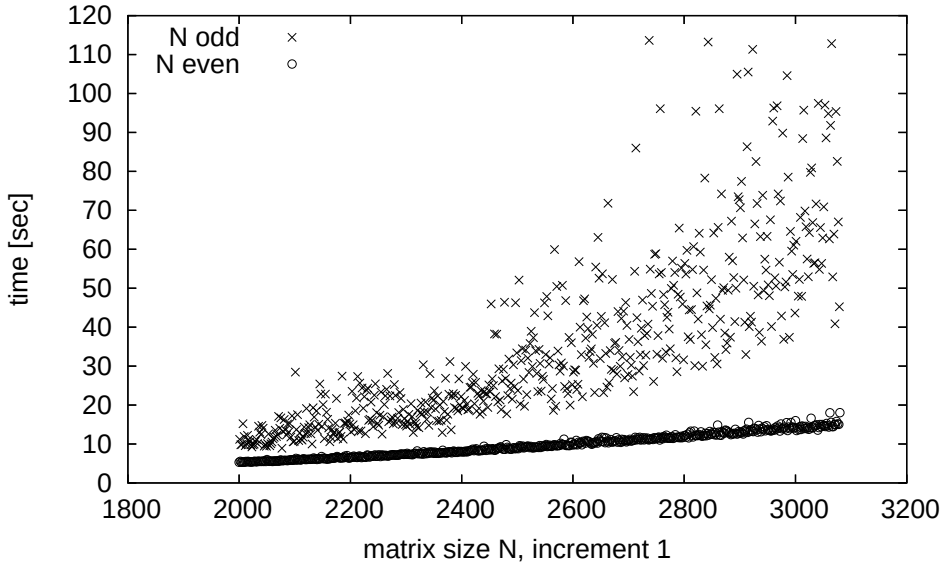


Figure 5: Long-run analysis: $P = 64$(VN), 8×8, $N = 2000, \ldots, 3080$, $NB = 20$

The first thing to notice is that the measurements are divided into odd and even, too. On both figures the odd values of N are on top of the plot with no exception, whereas the even values of N in Fig. 5 form a regular curve. This behavior changes by switching NB to 21. Fig. 6 shows that timings produced by the odd values remain on their positions, but now there are periodical interleaves concerning the even values. These interleaves overlap a range of approximately 168 values of N. Choosing a grid-shape of 8×8 the number 168 seems to be a reasonable magnitude as the product of 8 and NB. Tests with $NB = 41$ or $NB = 201$ show that these interleaves depend essentially on the chosen block size, e.g. interleave-range of approximately 320 at $NB = 41$. Another aspect of these results is that timings produced by the odd values increase with a higher speed than execution times generated by the even ones.
The unfavorable effect of the odd values depends basically on the special memory alignment of JUBL. To keep the second FPU busy and to get an optimal performance it is necessary to keep the data into the primary and secondary register. With the PPC440 hardware architecture it is allowed to load a quadword, i.e. 2 double precision variables, per cycle: 8 bytes in the primary register and 8 bytes in the corresponding secondary register [1]. If there is an odd number of columns and rows of the matrix or the blocks, this kind of memory alignment fails which leads to a delay. As a result it is recommended not to use odd matrix and block sizes in applications if this can be avoided.

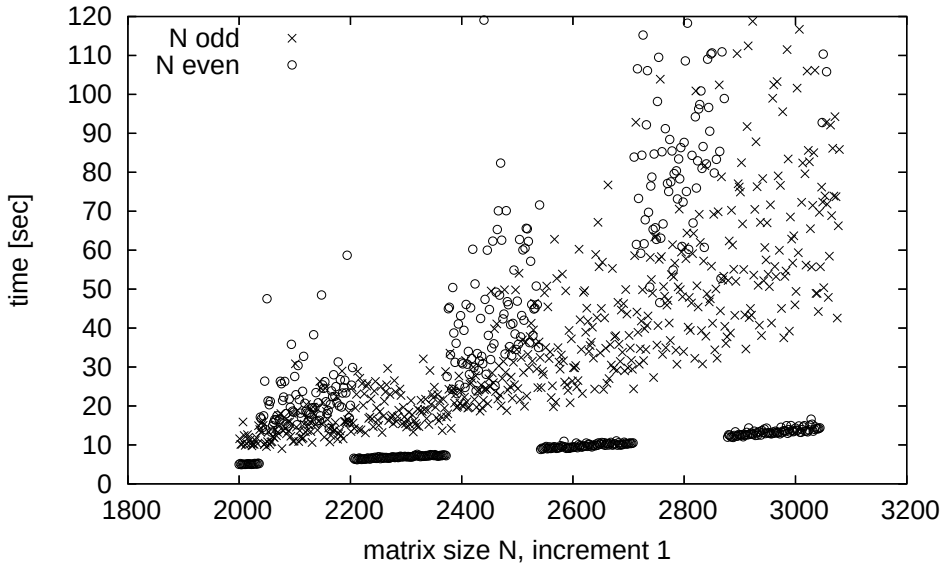


Figure 6: Long-run analysis: $P = 64$(VN), 8×8, $N = 2000, \ldots, 3080$, $NB = 21$

Scalability

Taking the conventions of the initialization each test is set up identically except for $P = 512$ and higher. The matrix size varies from $N = 1000$ to $N = 5000$ with increment 50, the block size is $NB = 20$ and the grid-shapes are natively chosen. The amount of processors doubles with each test, starting from $P = 32$ up to $P = 1024$(VN). Getting 512 and more processors means to switch to midplane 0 which limits the available time for these tests, so that they are set up with $N = 2000, \ldots, 5000$ and an increment of 100. The following figures give an overview of the results.

Fig. 7 compares six different numbers of processors recorded in a logscale with base 2. The difference between $P = 32$ and $P = 64$ is the largest one, whereas the gap between the two highest amounts of processors is only exiguous. Comparing $P = 1024$(VN) and $P = 32$ it is obvious that especially with a high number of processors no appreciable advantage can be achieved concerning large matrices (particularly with regard to $N = 4000$ and higher). The outliers can be marked again: Concerning $N = 3700$ and $N = 4700$ (which does not fit on the plots) these unusual structures occur on every plot except for $P = 512$ and $P = 1024$(VN).

The almost linear characteristic of the plot in Fig. 7 indicates an exponential increase in time. Instead of flattening as expected by the N^3 complexity of the algorithm, the curves become even steeper for larger N. The beginning of this increase depends on the number of used processors: If more processors are used, the curves loose their linear attitude earlier.

Fig. 8 shows the speedup curves. The speedup is computed relatively to $P = 32$ which is denoted by a constant curve at the bottom of the figure. As mentioned before the gap between 32 and 64 processors is as distinct as expected, but the distance between 512 and 1024(VN) processors is very small, which means no real improvement by doubling the number of processors. The ideal is a speedup of 2 if the number of processors doubles, which can be achieved only with 32 and 64 processors. Adding up the speedup should lead to a value of circa 32 by doubling the amount of processors 5 times from 32 to 1024, but these results show just a maximum speedup of 8.

Another remarkable aspect of these results is, that the speedup is very inconstant and irregular. Using a higher amount of processors, the speedup curve begins to decrease at lower values of N. This leads to

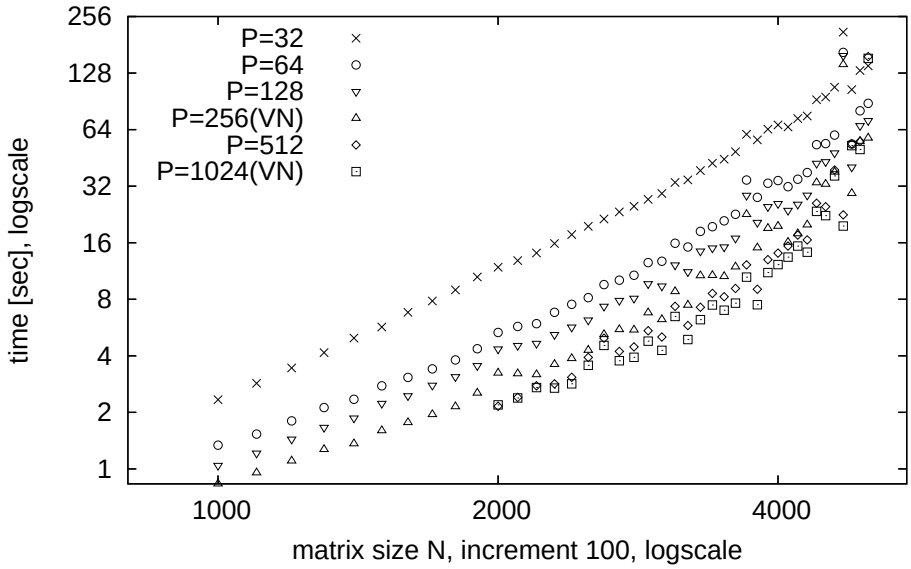


Figure 7: Comparison of processor numbers: $N = 1000, \ldots, 5000,\ NB = 20$ (logscale)

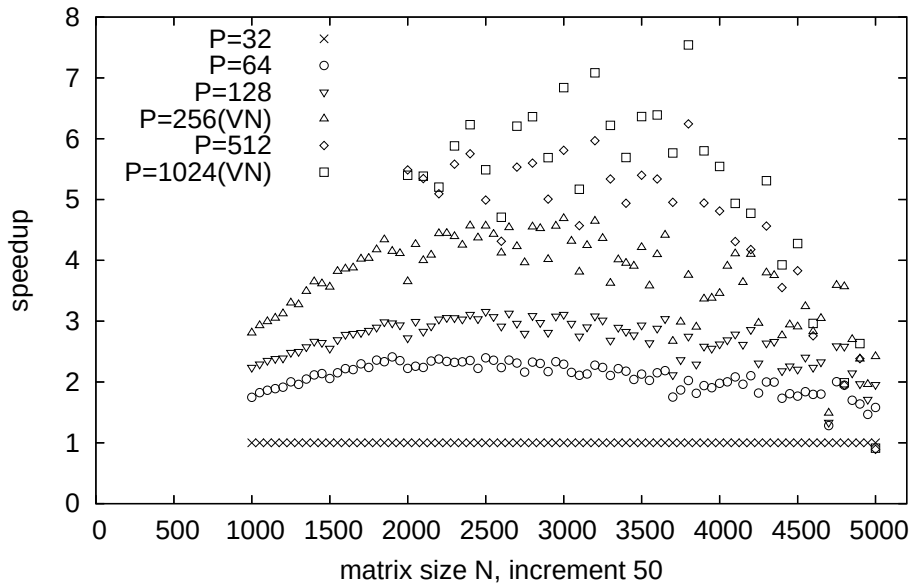


Figure 8: Scalability: $N = 1000, \ldots, 5000,\ NB = 20$

an unacceptable behavior concerning larger matrix sizes.
The speedup analysis shows clearly an urgent need of optimization. The isolated occurrence of outliers is basically caused by the structure of the testing program, but the large differences between the speedup ideal and the results has to be ascribed primarily to optimization and communication problems.

Review and Conclusion

Complications

One of the main problems of this project is the severely limited I/O-ability of JUBL. There are unpredictable I/O-failures which cause the whole program to crash. These failures can appear if a program tries to open, close, write or read a file during computation. The only reliable mechanism is the standard output of a program which seems to be not affected by this problem. Thus at the moment it is impossible to use profiling or tracing tools so that for a detailed analysis an instrumentation workaround is needed (see "JUMP vs. JUBL").
Another main problem is the total lack of optimization. As mentioned before both ScaLAPACK and the testing program have been compiled with `-O0` which means no optimization at all. This restriction is required, because flags like `-O2` produce unexpected failures, e.g. irregular program termination after using the grid-shape 4×16 whereas 2×32 produces no failure. Nevertheless compiling the program and ScaLAPACK with `-O2` leaves a positive mark. A first check with uncomplicated values shows a noticeable performance increase.
According to the unpredictable occurrence of peak points like $P = 32, N = 3700, NB = 20$ or $P = 16, N = 4112, NB = 23$ (see "JUMP vs. JUBL"), the structure of the testing program has some disadvantages. Both randomized starting vectors (eigenvalues and Householder reflector) are created at the beginning of each loop cycle. As a consequence e.g. the matrix with size $N = 3700$ constructed for the runs on up to 128 processors is not the same as the one constructed for runs with 512 and 1024(VN) processors, which means that the problem complexity is extensively controlled by the routine `RANDOM_NUMBER` and its restarts. This causes an interdependency of measurement and starting values and might be a reason for missing outliers if 512 or 1024(VN) processors are used. The section "Standardization of the input matrix" provides a method of solution for this problem.
Finally there is an accuracy difference between the measurements on JUMP and JUBL. Running the tests on JUBL the accuracy of the eigenvalues is close to $1.0E^{-10}$ whereas the results on JUMP return values in range of $1.0E^{-11}$. This might be a problem of optimization. Higher levels of optimization keep the computed values as long as possible in the registers which effects the accuracy essentially, because on JUMP the values kept in registers have a higher accuracy than those in memory.

JUMP vs. JUBL

After testing the program on JUBL it is essential to check its correctness on JUMP. The following results have to be interpreted with care, because the assumptions on JUMP and JUBL differ a lot, e.g. optimization level, communication routines or memory availability.
Fig. 9 shows a comparison of the execution times on JUMP and JUBL with 32 processors and $N = 1000, \ldots, 4000$. The first thing to notice is that the gap of the timings becomes larger with increasing N. Starting with a factor of 2 at $N = 1000$ JUMP is nearly 6 times faster than JUBL at $N = 4000$. Although this factor is too high the current major problem is its quick growth, normally there should be a constant speedup. Next to this difficulty it has to be mentioned, that the peak points and their structure at $N = 3700$ appear on JUMP again. This fact corroborates the argumentation that this is not an optimization problem but rather depends on the structure of the testing program.
To use a kind of profiling tool without I/O-failures the testing program and the routines were modified. The routines `PDSYEVX` and `PDSTEIN` were divided natively into different sections and returned next to their result the detailed timings for each of these parts. Taking for example the peak point $P = 16, N = 4112, NB = 23$ the following scheme gives an overview of the maximum time a processor needs for each section:

JUBL: N=4112, NB=23, Grid 4x4, max. time:	1777.27285873000005
Tests and Setup (MAX):	0.689794942857142879E-01
Call PDSYNTRD (MAX):	38.1730634271428571
Redist. tridiagonal matrix (MAX):	0.283804999999999995E-02
Call PDSTEBZ (MAX):	6.54889558999999988
Call PDSTEIN (MAX):	1720.71151298285713
Test and Setup (MAX):	0.197795000000000002E-02
Call DSTEIN2 (MAX):	1719.07637001571425
Redist. eigenvector matrix (MAX):	0.679070000000000043E-03
Apply permutation (MAX):	1.63241624714285716
Sort IWORK (MAX):	0.215428571428571429E-05
Call PDORMTR (MAX):	11.7674603599999994
Rescale eigenvalues (MAX):	0.117142857142857139E-06

The computation of this problem uses more than 1777 sec. on JUBL. This amount of time is needed just by one processor, whereas the others have to wait, which results in an extremly unbalanced behavior.
The routine `PDSYEVX` spends nearly 95% for the inverse iteration and reorthogonalization `PDSTEIN`. Dividing this routine into sections, too, shows that `DSTEIN2` needs most of the time. This distribution was verified with different tests. Nevertheless this is an unexpected result: According to its sequential character `DSTEIN2` is a pure computation routine so that this cannot be ascribed to communication trouble.
The next table shows the same problem computed on JUMP:

JUMP: N=4112, NB=23, Grid 4x4, max. time:	28.2871108055114746
Tests and Setup (MAX):	0.192475318908691406E-01
Call PDSYNTRD (MAX):	15.5302925109863281
Redist. tridiagonal matrix (MAX):	0.295042991638183594E-01
Call PDSTEBZ (MAX):	1.62795519828796387
Call PDSTEIN (MAX):	7.03478884696960449
Test and Setup (MAX):	0.843048095703125000E-03
Call DSTEIN2 (MAX):	6.96198463439941406
Redist. eigenvector matrix (MAX):	0.126123428344726562E-03
Apply permutation (MAX):	7.02518773078918457
Sort IWORK (MAX):	0.214576721191406250E-05
Call PDORMTR (MAX):	4.13338375091552734
Rescale eigenvalues (MAX):	0.190734863281250000E-05

Here the algorithm needs just about 25% of the time for the inverse iteration, but the call of `DSTEIN2` takes most of the time again. This leads to the conclusion that there must be a problem with the reorthogonalization or the convergence of the inverse iteration on JUBL, but this can only be ascribed partly to the lack of optimization.

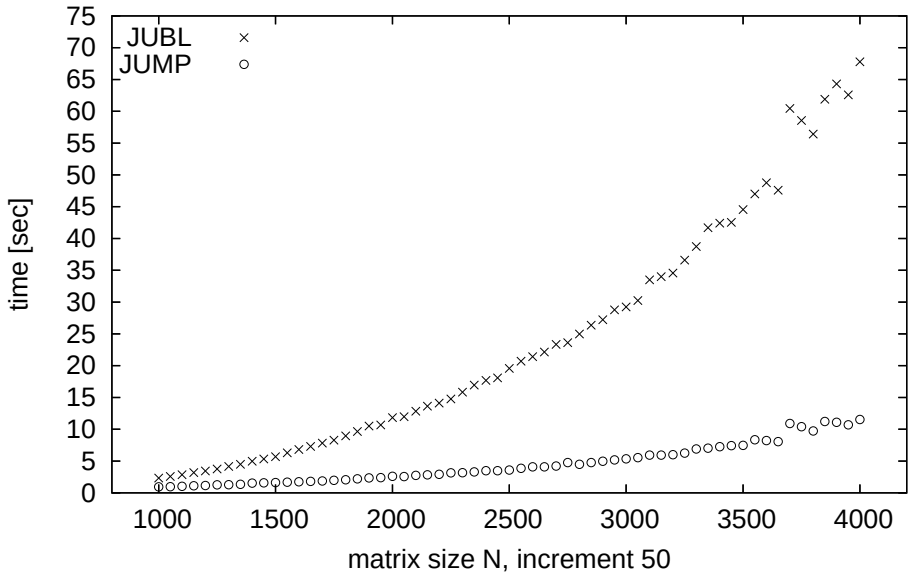


Figure 9: JUMP vs. JUBL: $P = 32,\ 8 \times 4,\ N = 1000, \ldots, 4000,\ NB = 20$

Standardization of the input matrix

To avoid the disadvantage of the testing program it is necessary to remove the creation of the randomized vectors from the loop and to initialize them globally at the beginning of the program. The improved version of the program continues after the initialization with the construction of max_N randomized and unsorted eigenvalues and a Householder vector with the dimension max_N so that for each loop cycle $N \leq max\_N$ is true. After entering the loop the program takes the first N eigenvalues out of the global eigenvalue vector and sorts them into a local array. The local reflector consists of the first N entries of the global Householder vector. The local arrays are deallocated at the end of each cycle. Using this modification it is guaranteed that the program uses alway the same set of values for the eigenvalues and the reflector, because `RANDOM_NUMBER` is called only at the beginning of the program, but not at each loop cycle. This improvement detaches the problem complexity and the starting values. The following results show the detailed progresses.
Fig. 10 repeats one of the scalability tests using $P = 64$(VN) and $N = 1000, \ldots, 5000$. The curve produced with the improved program is much smoother than the original one. This is an important indication of complexity harmonization. Just one noticeable anomaly located near $N = 4500$ interrupts the curve. The location of this point depends on the value max_N which influences the behavior of `RANDOM_NUMBER` at the beginning of the program. If max_N changes, the discontinuity is relocated, but there seems to be no comprehensible correlation.
In Fig. 11 the long-run analysis with $P = 64$(VN), $N = 2000, \ldots, 3080$ and a block size of 20 is repeated. The even values of N form nearly the same curve than before, but now the odd values behave more regularly which is a remarkable harmonization, too.
Although there are some outliers which may be caused by the value of max_N and an unfavorable distribution, the standardization of the input matrix is a great improvement.

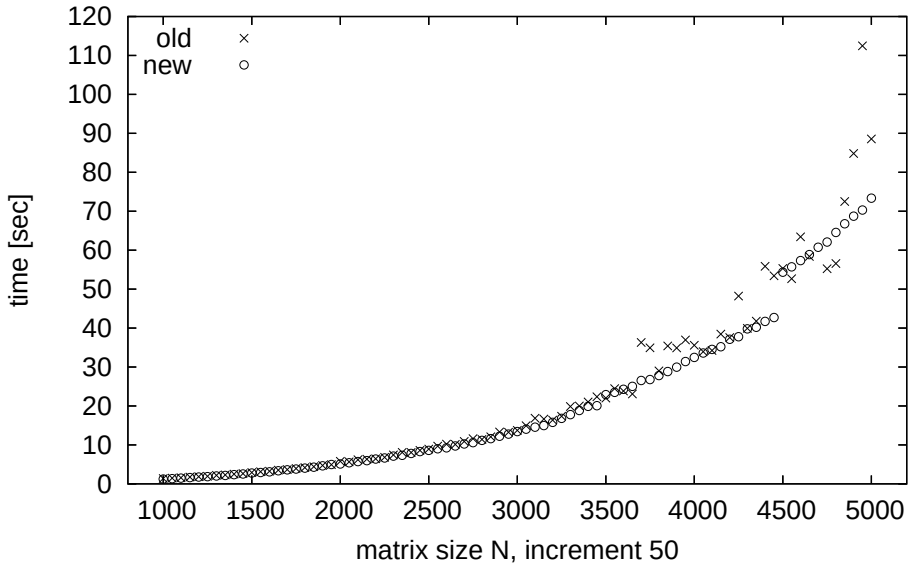


Figure 10: Compare program improvement: $P = 64$(VN), 8×8, $N = 1000, \ldots, 5000$, $NB = 20$

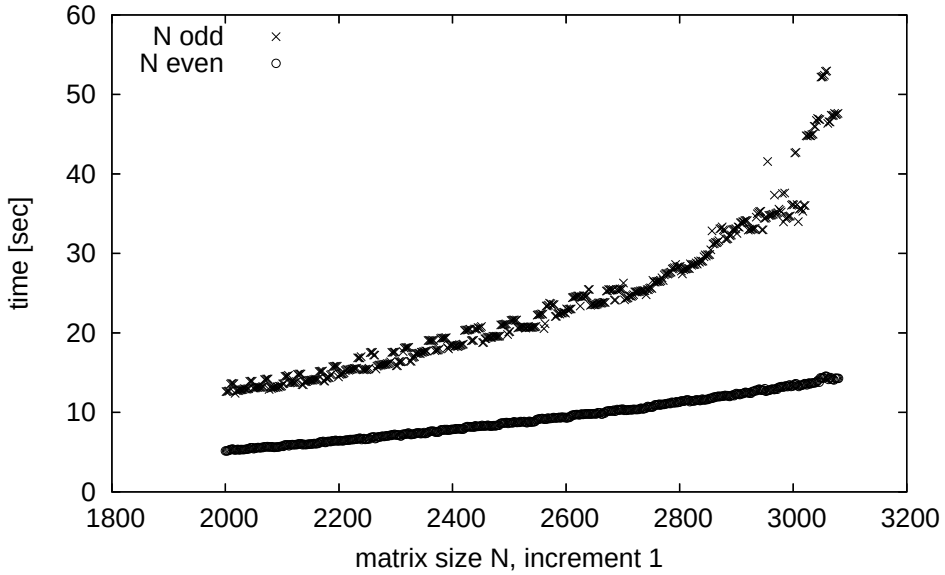


Figure 11: Long-run analysis (improved algorithm): $P = 64$(VN), 8×8, $N = 2000, \ldots, 3080$, $NB = 20$

Prospect

The aim of this project has been to analyze the performance of PDSYEVX on JUBL. Concerning the serious problems it is not possible to perform a full and significant analysis at the moment. The software on JUBL is still under development which causes many restrictions and compromises, e.g. concerning profiling / tracing or optimization. An important step towards a better performance analysis will be the

release of fundamental software packages like ESSL. With these releases a deeper analysis of the scalability and the remaining performance bottlenecks will be possible.
Some aspects of an entire performance analysis will have to be investigated with more in-depth. A major analysis of JUMP and JUBL's differences concerning timing and accuracy will lead to reference values JUBL has to match with, but these tests are not representative until all necessary optimization levels and software packages are available. Another aspect is the examination of communication problems in connection with different grid-shapes and their effects on the results, e.g. the recurring structure of peak points. Furthermore it would be useful to either eliminate the outliers completely or to make them more predictable. There is still no closed theory which can handle these occurrences.
Nevertheless a few conclusions can be drawn: The performance of the eigensolver PDSYEVX depends essentially on optimization and a sensible choice of the problem parameters. As seen at the long-run analysis an unfavorable set of values can cause unexpected behavior. The user is responsible for a consistent choice of parameters to avoid extended loss of time, even if JUBL reaches production status.

Acknowledgements

This project is a result of the Guest Students´ Programme "Scientific Computing" 2005, organised by ZAM and NIC. I would like to thank my supervisor I. Gutheil for her patient support and for helping me with all my questions and complaints. Finally I would like to thank Dr. R. Esser for enabling me to participate in this program.

References

1. Unfolding IBM eServer Blue Gene Solution, Redbook, September 2005
   http://www.fz-juelich.de/zam/docs/BlueGene/IBM-BGL-RB-unfolding.pdf
2. ZAM aktuell Nr. 136, Juli/August 2005
   http://www.fz-juelich.de/zam/docs/za/2005/za-136
3. First official release of BG/L software (V1R1M0)
   http://www.fz-juelich.de/zam/ibm-bgl/usage/FAQ/V1R1M0
4. IBM XL Fortran
   http://www-306.ibm.com/software/awdtools/fortran/xlfortran/
5. The Message Passing Interface (MPI) standard
   http://www-unix.mcs.anl.gov/mpi/index.htm
6. BLAS Frequently Asked Questions (FAQ)
   http://www.netlib.org/blas/faq.html
7. BLAS: dgemm.f
   www.netlib.org/blas/dgemm.f
8. BG/L Software / Libraries / Tools
   http://www.fz-juelich.de/zam/ibm-bgl/software
9. The BLACS Project
   http://www.netlib.org/blacs/
10. The ScaLAPACK Project
    http://www.netlib.org/scalapack/
11. I. Gutheil, R. Zimmermann
    Performance of Software for the Full Symmetric Eigenproblem on CRAY T3E and T90 Systems
    John von Neumann-Institut, Forschungszentrum Juelich, July 2000
12. L.S. Blackford, J. Choi, A.Cleary et al.
    ScaLAPACK Users´ Guide
    SIAM Philadelphia, 1997
    Also available via WWW under
    http://www.netlib.org/scalapack/slug/scalapack_slug.html

Arnoldi-Verfahren für Staggered Fermionen in der Gitter-Quantenchromodynamik

Aiko Voigt

Institut für Physik der Humboldt Universität zu Berlin

E-mail: aiko.voigt@physik.hu-berlin.de

Zusammenfassung: Ziel dieser Arbeit ist die numerische Bestimmung der Eigenwerte und Eigenvektoren der masselosen One Link Staggered Fermion Matrix, wie sie in numerischen Simulationen der Gitter-Quantenchromodynamik verwendet wird. Dazu wird das Fortran 77 Paket P_ARPACK genutzt. Dieses implementiert die Implicitly Restarted Arnoldi Methode (IRAM).

Einleitung

Die Physik kennt vier fundamentale Kräfte: die Gravitation, die elektromagnetische, die schwache sowie die starke Kraft. Bis auf die Gravitation können alle den Kräften zugrunde liegenden Wechselwirkungen durch sogenannte Quantenfeldtheorien im Rahmen des Standardmodells der Elementarteilchenphysik beschrieben werden. Dabei liefert das Glashow-Salam-Weinberg Modell die vereinigte Theorie der elektromagnetischen und schwachen und die Quantenchromodynamik (QCD) die Theorie der starken Wechselwirkung.

Die starke Wechselwirkung ist wesentlich für den Zusammenhalt der Atomkerne. Neutronen und Protonen als Bausteine der Atomkerne sind jedoch keine Elementarteilchen. Die elementaren Freiheitsgrade der QCD sind vielmehr die Quarks und die Gluonen. Die Gluonen vermitteln als Kraft-Austauschteilchen die starke oder auch Farb-Kraft zwischen den Quarks. Neutronen und Protonen oder z.B. die Mesonen wie das π-Meson sind Bindungszustände aus Quarks und Gluonen.

In der QCD tragen neben den Quarks auch die Gluonen eine Farb-Ladung. Hier unterscheiden sich die Gluonen von den Kraft-Austauschteilchen der elektromagnetischen Kraft, den Photonen, welche keine elektrische Ladung tragen. Als Konsequenz wechselwirken die Gluonen direkt miteinander und tragen so wesentlich etwa zu der π-Masse bei. Diese Eigenschaft der Gluonen stellt sich als entscheidende Komplikation bei der Analyse der Theorie im physikalisch relevanten Bereich kleiner Impulse heraus.

Die einzig bekannte Herangehensweise an die Theorie in diesem Bereich der Impulse, die eine weitere Modellbildung vermeidet, ist die Gitter-QCD. Dabei wird die Theorie auf ein euklidisches Raumzeit-Gitter abgebildet. Die so diskretisierte Theorie liefert im Limes des verschwindenden Gitterabstands die ursprünglich im Kontinuum formulierte Theorie zurück. Die Diskretisierung ist dabei keineswegs eindeutig, da sie nur für verschwindenden Gitterabstand durch die Kontinuumstheorie vorgegeben ist. Entsprechend existieren verschiedene Ansätze die Wirkung der QCD, die allgemein für jede Quantenfeldtheorie zentrale Größe, auf das Gitter abzubilden.

In dieser Arbeit werden Rechnungen mit einem Anteil der auf spezielle Art diskretisierten Wirkung der QCD durchgeführt. Dieser Anteil enthält die Wechselwirkung der Quarks mit den Gluonen und den Quarkmassen-Term und wird, da Quarks Fermionen sind, auch als fermionischer Anteil der Wirkung bezeichnet. Im Rahmen von numerischen Simulationen der Gitter-QCD wird er mit Hilfe der sogenannten

Fermion- oder Dirac-Matrix implementiert.

Bei einer numerischen Simulation der Gitter-QCD werden die Eigenwerte und Eigenvektoren der Dirac-Matrix über verschiedene numerische Verfahren indirekt oder aber auch direkt für die Berechnung von physikalischen Größen benötigt. Es ist daher von Interesse, ein effizientes Verfahren zur Bestimmung der Eigenwerte und Eigenvektoren der Dirac-Matrix für die gewählte Diskretisierung zur Verfügung zu haben.

Ziel dieser Arbeit ist die Bereitstellung eines solchen Verfahrens für die Dirac-Matrix der auf Kogut und Süsskind zurückgehenden Diskretisierung des fermionischen Anteils der Wirkung der Gitter-QCD.

Quarks auf dem Gitter

Eine Einführung in die Gitter-QCD findet sich in [1].

Das Verdopplungsproblem für Fermionen auf dem Gitter

Freie fermionische Teilchen wie etwa Quarks ohne Farbladungen werden durch die Diracgleichung beschrieben

$$(i\gamma^\mu \partial_\mu - M)\psi\,(x) = 0\,. \tag{1}$$

Dabei wird der Einsteinschen Summenkonvention gemäß über $\mu = 0, ..., 3$ summiert. M bezeichnet die Fermionenmasse und die γ^μ die 4×4 Dirac-Matrizen und genügen der Antikommutatorrelation

$$\{\gamma^\mu, \gamma^\nu\} = 2g^{\mu\nu} \tag{2}$$

und ψ ist das 4-komponentige Spinorfeld. Hier und im Folgenden sind die Spinorindizes in der Regel unterdrückt. Die Fermionische Wirkung der freien Theorie S_F ergibt sich mit $\bar{\psi} \equiv \psi^\dagger \gamma^0$ zu

$$S_F[\bar{\psi}, \psi] = \int d^4x \bar{\psi}(x)(i\gamma^\mu \partial_\mu - M)\psi\,(x)\,. \tag{3}$$

- Nach Übergang von der Minkoswki-Metrik zur Euklidischen Metrik durch Wickrotation der Zeitkoordinate $x^0 \rightarrow -ix^4$ erhält man für die freie fermionische Wirkung im Euklidischen

$$S_F^E[\bar{\psi}, \psi] = \int d^4x \bar{\psi}(x)(\gamma_\mu^E \partial_\mu + M)\psi\,(x)\,. \tag{4}$$

Im Weiteren wird immer die euklidische Metrik betrachtet, der Index E entfällt daher.

Im Gegensatz zu den Eichfeldern und den skalaren Felder der Theorie führt der Versuch, Fermionen auf das Gitter zu bringen zu Problemen. Geht man in analoger Weise zu den skalaren Feldern vor und schreibt

$$
\begin{aligned}
M &\rightarrow \frac{1}{a}\hat{M}, \\
\psi(x) &\rightarrow \frac{1}{a^{3/2}}\hat{\psi}(n), \\
\bar{\psi}(x) &\rightarrow \frac{1}{a^{3/2}}\bar{\hat{\psi}}(n), \\
\partial_\mu\psi(x) &\rightarrow \frac{1}{a^{5/2}}\hat{\partial}_\mu\hat{\psi}(n),
\end{aligned}
$$

wobei die diskretisierte Ableitung $\hat{\partial}_\mu$ gegeben ist als

$$\hat{\partial}_\mu\hat{\psi} = \frac{1}{2}[\hat{\psi}(n+\hat{\mu}) - \hat{\psi}(n-\hat{\mu})]. \tag{5}$$

erhält man als erste Form der Gitterwirkung freier Fermionen den Ausdruck

$$S_F^{naive} = \frac{1}{2}\sum_n \gamma_\mu \bar{\hat{\psi}}(n)\left[U_{n,\mu}\hat{\psi}(n+\hat{\mu}) - U^\dagger_{n-\hat{\mu},\mu}\hat{\psi}(n-\hat{\mu})\right] + \hat{M}\sum_n \bar{\hat{\psi}}(n)\hat{\psi}(n). \tag{6}$$

Dabei bezeichnet $\hat{\mu}$ den in μ-Richtung zeigenden Gittervektor der Länge des Gitterabstandes a.

Einer Theorie mit Wechselwirkung erhält man aus der freien Theorie durch Einbeziehung der Linkvariablen $U_{n,\mu} \in SU(3)$. Die „naive" volle fermionische Wirkung auf dem Gitter ergibt sich damit zu

$$S_F^{naive} = \frac{1}{2}\sum_n \gamma_\mu \bar{\hat{\psi}}(n)\left[U_{n,\mu}\hat{\psi}(n+\hat{\mu}) - U^\dagger_{n-\hat{\mu},\mu}\hat{\psi}(n-\hat{\mu})\right] + \hat{M}\sum_n \bar{\hat{\psi}}(n)\hat{\psi}(n). \tag{7}$$

Die Wirkung (7) erhält die chirale Symmetrie der kontinuierlichen Theorie. Allerdings führt sie zum Verdopplungsproblem der Fermionen, sodass man 16 anstatt nur einem Fermion erhält. Nielsen und Ninomiya [2] konnten zeigen, dass es nicht möglich ist eine lokale, translationsinvariante, hermitesche Wirkung für Fermionen auf dem Gitter zu definieren, die die chirale Symmetrie im Limes $M \rightarrow 0$ erhält und keine Doppler aufweist.

Staggered Fermionen

Ein auf Kogut und Süskind zurückgehender Ansatz das Doppler-Problem zu umgehen sind die sogenannten Staggered Fermionen. Diese verletzen die Lokalität der Wirkung, erhalten aber einen Rest der chiralen Symmetrie des Kontinuums. Das Verdopplungsproblem wird bei Staggered Fermionen um den Faktor 4 reduziert. Somit beschreibt dieser Ansatz 4 massenentartete Fermionen.

Staggered Fermionen entstehen durch eine unitäre Transformation der Fermionenfelder $\hat{\psi}$ beziehungsweise $\bar{\hat{\psi}}$

$$
\begin{aligned}
\hat{\psi}(n) &= T(n)\chi(n), \\
\bar{\hat{\psi}}(n) &= \bar{\chi}(n)T^\dagger(n).
\end{aligned}
$$

Die $T(n)$ sind unitäre 4×4 Matrizen, die so gewählt werden, daß

$$T^{\dagger}(n)\gamma_{\mu}T(n+\hat{\mu}) = \eta_{\mu}(n) \tag{8}$$

gilt. Die c-Zahlen η_{μ} werden als Kogut-Süskind-Phasen bezeichnet. Konkret kann diese Transformation durch

$$T(n) = \gamma_1^{n_1}\gamma_2^{n_2}\gamma_3^{n_3}\gamma_4^{n_4} \tag{9}$$

realisiert werden. Die Kogut-Süskind-Phasen sind dann gegeben durch

$$\eta_{\mu}(n) = (-1)^{\sum_{\nu<\mu} n_{\nu}}. \tag{10}$$

Die fermionische Wirkung, geschrieben in den transformierten Spinorfeldern χ und $\bar{\chi}$, nimmt nun folgende Form an

$$S_F^{stag} = \sum_{n,\mu,\alpha} \eta_{\mu}(n)\bar{\chi}_{\alpha}(n)[U_{n,\mu}\chi_{\alpha}(n+\hat{\mu}) - U^{\dagger}_{n-\hat{\mu},\mu}\chi_{\alpha}(n-\hat{\mu})] + \hat{M}\sum_{n}\bar{\chi}_{\alpha}(n)\chi_{\alpha}(n). \tag{11}$$

Die Dirac-Matrizen γ_{μ} erscheinen nicht explizit in dieser Formulierung der Wirkung. Die Summation über den Spinorindex α kann nun im Prinzip von $\alpha = 1, ..., k$ erfolgen. Wählt man $k = 1$, so folgt

$$S_F^{stag} = \frac{1}{2}\sum_{n,\mu} \eta_{\mu}(n)\,\bar{\chi}(n)\left[U_{n,\mu}\chi(n+\hat{\mu}) - U^{\dagger}_{n-\hat{\mu},\mu}\chi(n-\hat{\mu})\right] + \hat{M}\sum_{n}\bar{\chi}(n)\,\chi(n)\,. \tag{12}$$

Der one link Staggered Fermionen Operator ergibt sich also zu

$$D_{n,m}^{stag} = \frac{1}{2}\sum_{\mu} \eta_{\mu}(n)\left[U_{n,\mu}\delta_{n+\hat{\mu},m} - U^{\dagger}_{n-\hat{\mu},\mu}\delta_{n-\hat{\mu},m}\right] + \hat{M}\delta_{n,m} \tag{13}$$

mit einem antihermiteschen und einem hermiteschen Anteil. Für Untersuchungen des Spektrums von (13) wird nur der erste antihermitesche Teil betrachtet, da die Eigenwerte durch den zweiten Summanden lediglich um den konstanten Wert $\hat{M}$ auf der reellen Achse verschoben werden. Im Folgenden sollen also die Eigenwerte von

$$D_{n,m}^{stag} = \frac{1}{2}\sum_{\mu} \eta_{\mu}(n)\left[U_{n,\mu}\delta_{n+\hat{\mu},m} - U^{\dagger}_{n-\hat{\mu},\mu}\delta_{n-\hat{\mu},m}\right] \tag{14}$$

berechnet werden. Das Spektrum des Operators D^{stag} ist aufgrund der Antihermitizität von (14) rein imaginär. Außerdem sind die Eigenwerte von (14) wie im Kontinuum symmetrisch zur reellen Achse [3]. Für typische Gitterdimensionen wird der Staggered Fermionen Operator beschrieben durch eine komplexe Matrix der Dimension $O(10^6)$. Damit ist klar, dass direkte Verfahren wie die QR-Zerlegung, die das gesamte Spektrum berechnen, aus Gründen der Rechenzeit und Speicherkapazität nicht anwendbar sind. Da man aber aus physikalischer Sicht besonders an den nahe bei Null liegenden Eigenwerten von (14) interessiert ist, bietet sich das Arnoldi-Verfahren an. Dieses und das Fortran 77 Paket P_ARPACK werden im nächsten Abschnitt näher beschrieben.

Arnoldi-Verfahren und P_ARPACK

Grundzüge des Arnoldi-Verfahrens

Das Arnoldi-Verfahren wurde 1951 von W. E. Arnoldi vorgeschlagen als direktes Verfahren, eine allgemeine Matrix auf obere Hessenberg Form zu bringen. Später stellte sich heraus, daß es auch als ein iteratives Verfahren zur näherungsweisen Bestimmung einiger Eigenwerte und zugehöriger Eigenvektoren eingesetzt werden kann. Der Algorithmus ist besonders geeignet für große schwach besetzte Matrizen. Hier ist der Aufwand für eine Matrix-Vektor-Multiplikation vergleichsweise gering.

Gegeben sei eine Matrix $A \in \mathbb{C}^{n \times n}$, deren Spektrum betrachtet werden soll, und ein Startvektor $q \in \mathbb{C}^n$. Der Krylov-Unterraum $\mathcal{K}(A, q, k)$ ist definiert als

$$\mathcal{K}(A, q, k) = \text{span}\{q, Aq, A^2q, \dots\ A^{k-1}q\}.$$

Im Zuge des Arnoldi-Verfahrens wird die Matrix A orthogonal auf den Unterraum $\mathcal{K}(A, q, k)$ projiziert. Die Eigenwerte der dabei entstehenden Matrizen H_k der Dimension k sind dann Näherungen für die extremen Eigenwerte der Matrix A, und zwar umso bessere Näherungen, je höher die Dimension der Krylov-Unterraums $\mathcal{K}(A, q, k)$ ist.

Eine besondere Variante des Arnoldi-Verfahrens ist die Implicitly Restarted Arnoldi Methode (IRAM). Dabei werden die üblicherweise bei Arnoldi-Algorithmen auftretenden Speicherplatzprobleme vermieden. IRAM ist in der Lage k Eigenwerte mit benutzerdefinierten Eigenschaften (größter Realteil, kleinster Absolutbetrag, ...) unter Verwendung eines Speicherplatzbedarfs von $2nk + O(k^2)$ ohne zusätzlichen Hilfsspeicher zu berechnen.

Weiterführende Informationen zum Arnoldi-Verfahren und der IRAM-Variante finden sich in [4].

P_ARPACK

P_ARPACK ist die parallele Version der Fortran77 Bibliothek ARPACK für distributed memory - Parallelrechner. ARPACK benutzt eine algorithmische Variante des Arnoldi Prozesses, die Implicitly Restarted Arnoldi Method (IRAM). Für diese Arbeit sind die folgenden Merkmale von (P_)ARPACK von Bedeutung:

- (P_)ARPACK berechnet Eigenwerte am Rand des Spektrums. Die Zahl der zu berechnenden Eigenwerte und der relevante Teil des Spektrums (größter Realteil, kleinster Absolutbetrag, ...) wird vom Benutzer festgelegt.
- Die zugehörigen Eigenvektoren können auf Wunsch ebenfalls berechnet werden.
- Entartete Eigenwerte stellen kein Hindernis dar.
- Die Matrix, deren Eigenwerte berechnet werden sollen, muss nicht explizit vorliegen. Der Benutzer muss lediglich eine Routine zur Matrix-Vektor-Multiplikation bereitstellen.

Eine Besonderheit von P_ARPACK ist das Reverse Communication Interface. Es wird typischerweise wie folgt benutzt:

```
reverse_comm:
_naupd(..,&ido,&bmat,...,which,&nev,&tol,&ncv,...);
if(ido == -1 || ido == 1){
```

```
matrixvec(...);
goto reverse_comm;}
```

`ido` ist der reverse communication Parameter, der entscheidet, ob erneut ein Aufruf der _naupd Routine und eine Matrix-Vektor-Multiplikation nötig sind. Über `bmat` teilt der Benutzer mit, ob ein Standardeigenwertproblem oder ein generalisiertes Eigenwertproblem vorliegt. `which` spezifiziert den zu betrachtenden Teil des Spektrums, `nev` die Zahl der gewünschten Eigenwerte, `tol` die Genauigkeit und `ncv` die Anzahl der Arnoldivektoren. Nach dem diese Schleife verlassen wurde, wird einmal die Routine _neupd aufgerufen.

Darüberhinaus bietet (P_)ARPACK Routinen für Bandmatrizen und für die Singulärwertzerlegung.

Das Programm

Ziel des Projektes war das Erstellen eines Programmes, das auf einer einzulesenden Linkkonfiguration einen vom Benutzer zu spezifizierenden Teil des Spektrums des Staggered Fermionen Operators berechnet. Dazu wurden Teile des im Internet frei erhältliche und von der MIMD Lattice Computation (MILC) Collaboration entwickelten "MILC"-Code in der Version 6 (20.9.2002) benutzt [5]. Dieser liefert unter anderem die dem P_ARPACK bereitzustellende Matrix-Vektor-Multiplikation. Das Programm nutzt die P_ARPACK Routinen `pcnaupd` und `pcneupd` (`p` → parallel, `c` → single complex). P_ARPACK wurde im Standard Modus betrieben.

Woher kommen die Linkkonfigurationen?

Die Eigenwerte des Staggered Fermionen Operators werden auf einem Eichfeld-Hintergrund berechnet. Generell bietet der MILC-Code die Möglichkeit, Linkkonfigurationen für Staggered Fermionen zu erzeugen. Aus Gründen der Rechenzeit wurde auch auf Konfigurationen der Gauge Connection zurückgegriffen [6]. Die Gauge Connection ist ein Archiv für die Gitter-Quantenchromodynamik und stellt im Internet frei erhältlich Konfigurationen, die unter anderem auch von der MILC Kollaboration erzeugt wurden, bereit, insbesondere für Staggered Fermionen.

Ergebnisse

In den Berechnungen wurden zwei Konfigurationen verwendet. Diese sind in Tabelle 1 aufgeführt.

	u_MILC_1241.0250	88816_4.ascii
Nick	Konf_1212124	Konf_88816
n_x	12	8
n_y	12	8
n_z	12	8
n_t	4	16
N_F	2	4
β	5.25	5.40
$\hat{M}$	0.008	0.01

Tabelle 1: Verwendete Konfigurationen

Konf_1212124 enstammt dem Archiv der Gauge Connection. Diese Konfiguration beschreibt mittels „Wurzeltrick" N_F = 2 Flavours von Fermionen. Konf_88816 wurde mit Hilfe des MILC-Codes in einer Hybrid Monte Carlo Simulation gewonnen.

Es wurden immer `ncv` = 16 Eigenwerte angefordert und die Genauigkeit `tol` auf Maschinengenauigkeit gesetzt. Für die Zahl der Arnoldivektoren gilt `ncv` = 30. Für die Berechnungen wurde der Jülicher Supercomputer JUMP eingesetzt. Die Zahl der verwendeten Prozessoren ist stets 32.

Direkte Berechnung der tiefliegenden Eigenwerte

Die tiefliegenden Eigenwerte können direkt berechnet werden. Dazu sucht man nach den Eigenwerten mit geringstem Absolutbetrag (`which = 'SM'`).

Falten des Spektrums

Die Berechnung der Eigenwerte konnte erheblich beschleunigt werden, indem das Spektrum von $(D^{stag})^2$ betrachtet wurde statt die Eigenwerte direkt zu berechnen. Die Eigenwerte von $(D^{stag})^2$ sind reell und negativ. Für $(D^{stag})^2$ muss `which = 'LR'` gesetzt werden.

Die Eigenwerte des Operators D^{stag} können aus den Eigenwerten von $(D^{stag})^2$ rekonstruiert werden. Da das Spektrum von D^{stag} rein imaginär und symmetrisch zur reellen Achse ist, müssen die Eigenwerte von $(D^{stag})^2$ zweifach entartet sein. Sei $-\lambda^2$ mit $\lambda \in \mathbb{R}^+$ ein zweifach entarteter Eigenwert von $(D^{stag})^2$ mit den Eigenvektoren v_1 und v_2. Dann sind die Eigenwerte der komplexen 2×2-Matrix H

$$H_{ij} = \langle D^{stag} v_i | v_j \rangle \tag{15}$$

die gesuchten Eigenwerte $i\lambda$ und $-i\lambda$ von D^{stag}. Die Eigenwerte von H wurden mit der LAPACK-Routine cgeev berechnet.

Vergleich von direkter Berechnung der Eigenwerte und Berechnung über das gefaltete Spektrum

Die Ergebnisse sind in den Abbildungen 1 und 2 dargestellt.

Die Berechnung über das gefaltete Spektrum von $(D^{stag})^2$ war um den Faktor 3 (Konf_1212124) beziehungsweise 5 (Konf_88816) schneller als das direkte Verfahren, wie die in Tabelle 2 und 3 aufgeführten Werte zeigen. Der entscheidende Vorteil lag in der Zeitersparnis in der `pcnaupd` - Routine und in der geringeren Anzahl der benötigten Matrix-Vektormultiplikationen. Der Aufwand für die Berechnung der Eigenwerte von D^{stag} aus den Eigenwerten von $(D^{stag})^2$ kann vernachlässigt werden.

	direkt	Falten des Spektrums
Zeit in `pcnaupd` (s)	1351	429
Zeit in `pcneupd` (s)	0,015	0.003
Zeit für Matrix-Vektor-Multiplikationen (s)	417	216
Anzahl der Matrix-Vektor-Multiplikationen	695430	214605
Gesamtzeit (s)	1766	646

Tabelle 2: Vergleich der beiden Methoden zur Eigenwertberechnung für Konf_1212124

Darüberhinaus erzielte die Berechnung über die Faltung des Spektrums bessere Ergebnisse. Dies wird an der geringeren Abweichung der auf diesem Weg berechneten Eigenwerte von der imaginären Achse deutlich.

	direkt	Falten des Spektrums
Zeit in pcnaupd (s)	91,0	16,1
Zeit in pcneupd (s)	0,024	0,018
Zeit für Matrix-Vektor-Multiplikationen (s)	32,0	9,7
Anzahl der Matrix-Vektor-Multiplikationen	45932	6637
Gesamtzeit (s)	123,2	25,8

Tabelle 3: Vergleich der beiden Methoden zur Eigenwertberechnung für Konf_88816

Die Symmetrie der Eigenwerte zur reellen Achse war bei beiden Konfigurationen mit den zwei verwendeten Methoden erkennbar.

Die im Kontinuumslimes zu erwartende vierfache Entartung des Spektrums für die Konfiguration Konf_88816, beziehungsweise die zweifache Entartung der Eigenwerte für Konf_1212124, konnte nicht beobachtet werden. Dies konnte bei dem in dieser Arbeit verwendeten one link Staggered Fermionen Operator (14), den Werten für die Kopplungskonstante β und der Gittergeometrie jedoch auch nicht erwartet werden [3].

Kritisch anzumerken ist, daß das Programm mit einfacher Genauigkeit rechnet. Die verwendete Version des MILC-Codes machte dies leider notwendig. Bessere Ergebnisse wären durch Verwendung der P_ARPACK-Routinen für doppelte Genauigkeit pznaupd und pzneupd (z $\rightarrow$ double complex) möglich.

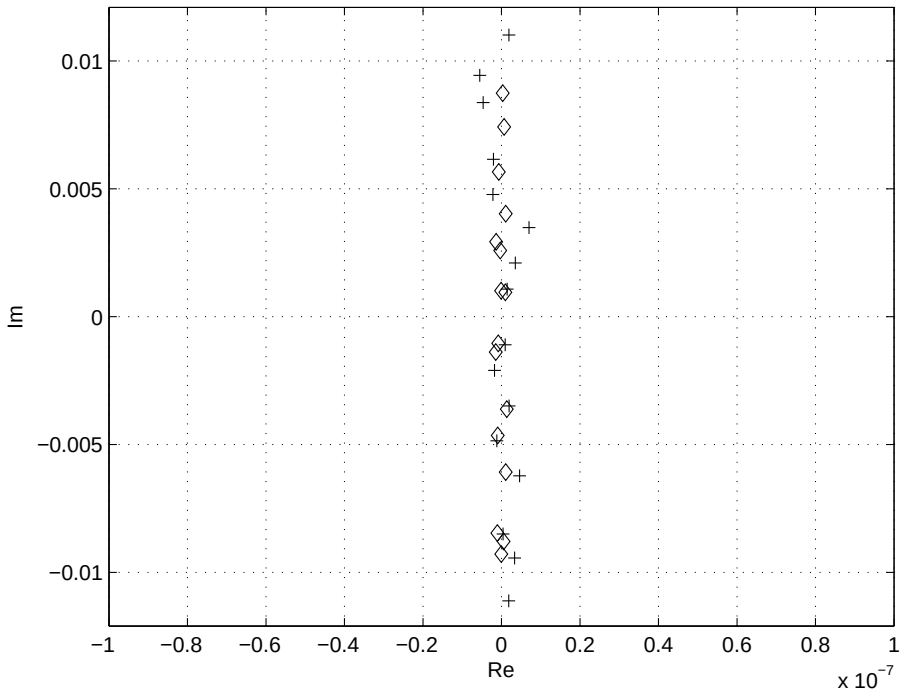


Abbildung 1: Die 16 kleinsten Eigenwerte für die Konfiguration Konf_1212124. Die Ergebnisse aus der direkten Berechnung sind mit + symboliert, die Ergebnisse aus der Faltung des Spektrums mit $\diamond$.

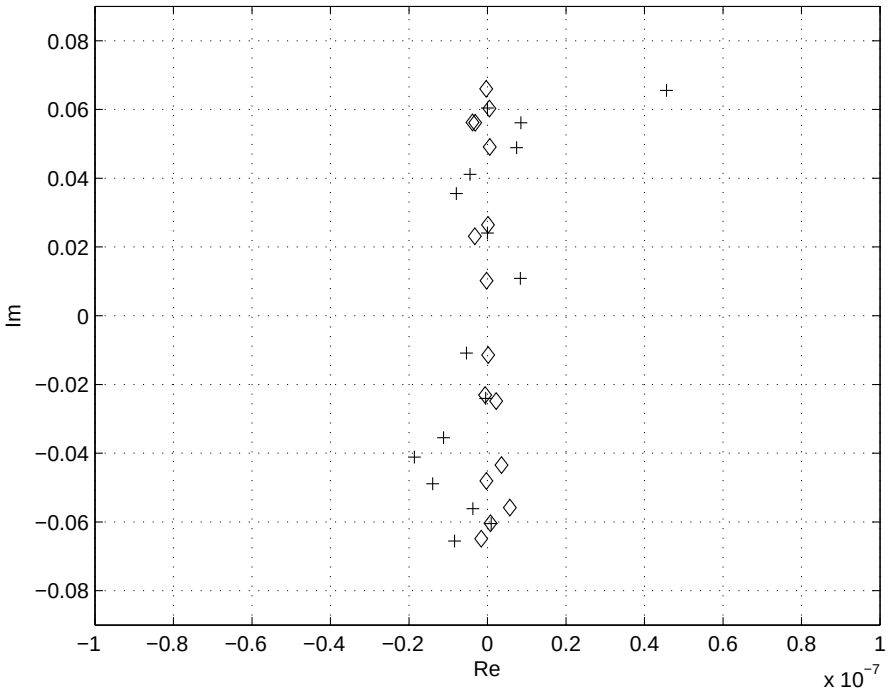


Abbildung 2: wie in Abbildung 1, jedoch für die Konfiguration Konf_88816

Fazit

Die tiefliegenden Eigenwerte des masselosen Staggered Fermionen Operators (14) konnten mit dem Paket P_ARPACK erfolgreich bestimmt werden. Dabei hat sich gezeigt, daß die Berechnung der Eigenwerte nicht direkt sondern über Faltung des Spektrums geschehen sollte.

Danksagung

Hiermit möchte ich mich bei meinem Betreuer Stefan Krieg bedanken. Desweiteren gilt Herrn Dr. Boris Orth mein Dank. Dem NIC, insbesondere Herrn Dr. Rüdiger Esser, sei für die Durchführung des Gaststudentenprogramms gedankt. Außerdem danke ich Herrn Professor Michael Müller-Preußker, der mir die Teilnahme an diesem Gaststudierendenprogramm ermöglicht hat.

Schlussendlich geht ein großer Dank an die anderen GaststudentInnen. Meinem geduldigen Zimmerkollegen Jens Grieger und dem niemals ermüdenden Tobias Hertkorn gebührt besonderer Dank.

Literatur

1. H. J. Rothe, Lattice Gauge Theories, World Scientific
2. H. B. Nielsen, M. Ninomiya, Nucl. Physics. B185 (1981) 20; Nucl. Phys. B193 (1981) 173
3. E. Follana, A. Hart, C. T. H. Davies, 2004 [hep-lat/0406010]
4. D. C. Sorensen, Implicitly Restarted Arnoldi/Lanczos Methods for Large Scale Eigenvalue Calculations, Parallel numerical algorithms, 1995
5. MILC: http://www.physics.utah.edu/∼detar/milc/
6. The Gauge Connection: http://qcd.nersc.gov/